МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЧОРНОМОРСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІМЕНІ ПЕТРА МОГИЛИ

Факультет комп’ютерних наук

Кафедра інженерії програмного забезпечення

**КОНСПЕКТ ЛЕКЦІЙ**

**з дисципліни «Системне програмування»**

для підготовки студентів

Напрям підготовки: 6.050101 – Комп’ютерні науки

Галузь знань: 0501 «Інформатика та обчислювальна техніка»

Освітньо-кваліфікаційний рівень: бакалавр

Укладач: к.т.н., ст. викл. Г.В. Горбань

Розглянуто на засіданні кафедри *інженерії програмного забезпечення*

Протокол №\_\_\_ від «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 201\_ р.

Миколаїв – 201\_

# Лекція №1. «Інструменти для автоматизації роботи в Microsoft Windows. Основи роботи в оболонці командного рядка Windows»

## Еволюція інструментів для автоматизації роботи в Microsoft Windows

В даний час графічний інтерфейс Windows став настільки звичним, що багато користувачів і адміністратори-початківці навіть не замислюються (а часто і просто не знають) про альтернативні способи управління цією операційною системою, пов'язані з командним рядком (command line) і різними сценаріями (scripts), про ті переваги, які дають ці інструменти з точки зору автоматизації роботи, тобто вирішення різних завдань в автоматичному режимі, без участі людини. Подібна ситуація обумовлена ​​тим, що історично командний рядок завжди був слабким місцем операційної системи Windows (в порівнянні з UNIX-системами). Причиною цього, в основному, є те, що спочатку компанія Microsoft орієнтувалася на широку аудиторію недосвідчених користувачів, які не бажають особливо вникати в технічні деталі виконання тих чи інших дій в системі. Тому основні зусилля розробників операційної системи прямували на поліпшення графічної оболонки для більш комфортної роботи непрофесіоналів, а не на створення робочого середовища для фахівців або досвідчених користувачів.

Як показав час, з комерційної точки зору на ринку персональних (домашніх або офісних) комп'ютерів ця стратегія виявилася більш ніж успішною - мільйони людей використовують графічний інтерфейс Windows для запуску потрібних їм програм, роботи в офісних пакетах, перегляду фільмів і т.п. Та й управляти одним Windows-сервером сьогодні нескладно - операційна система пропонує зручні графічні засоби для налаштування різних параметрів та виконання щоденних адміністраторських завдань, а за допомогою служби терміналів легко можна працювати на віддаленому сервері, фізично розташованому на іншому континенті.

Однак подібна модель управління не є масштабованою: якщо адмініструвати не один, а десять серверів, використовуючи стандартні графічні інструменти, то одну і ту ж послідовність зміни елементів управління в діалогових вікнах доведеться повторити десять разів, отже, в цьому випадку гостро постає питання про автоматизацію виконання рутинних операцій (наприклад, проведення інвентаризації обладнання та програмного забезпечення, моніторинг роботи служб, аналіз журналів подій і т.д.) на безлічі комп'ютерів. Допомогти в цьому можуть або спеціальні (як правило великовагові і недешеві) додатки типу Microsoft Systems Management Server (SMS), або сценарії, які пишуться адміністраторами самостійно (на мові оболонки командного рядка або на спеціальних мовах сценаріїв) і підтримуються безпосередньо операційною системою, без встановлення сторонніх програмних продуктів.

Тому для професіонала, що займається адмініструванням інформаційних систем на базі Windows, знання можливостей командного рядка, сценаріїв і технологій автоматизації, підтримуваних даною операційною системою, просто необхідно.

При цьому, однак, неправильно було б думати, що командний рядок або сценарії потрібні тільки адміністраторам. Адже щоденні рутинні завдання користувачів (пов'язані, наприклад, з копіюванням або архівуванням файлів, підключенням або відключенням мережевих ресурсів і т.п.), які зазвичай виконуються за допомогою графічного інтерфейсу провідника Windows, можна повністю самостійно автоматизувати, написавши нехитрий командний файл, що складається всього з кількох рядків. Більш того, для людини, яка не знає основні команди Windows і такі базові можливості операційної системи, як перенаправлення введення/виведення і конвейеризація команд, деякі найпростіші завдання можуть здатися нетривіальними. Спробуйте, наприклад, користуючись тільки графічними засобами, сформувати файл, який містить імена файлів з усіх підкаталогів будь-якого каталогу. Але ж для цього досить виконати єдину команду DIR (з певними ключами) і перенаправити виведення цієї команди в потрібний текстовий файл.

Яким же нам хотілося б бачити інструмент для автоматизації роботи в операційній системі, якими можливостями він повинен володіти? Бажано, щоб в ньому було реалізовано наступне:

* робота в різних версіях операційної системи (бажано у всіх) без встановлення будь-якого додаткового програмного забезпечення;
* інтеграція з командним рядком (безпосереднє виконання вводиться з клавіатури команд);
* узгоджений і сумісний синтаксис команд і утиліт;
* наявність докладної вбудованої довідки по командам з прикладами використання;
* можливість виконання сценаріїв, складених на простій для вивчення мові;
* можливість використання всіх технологій, підтримуваних операційною системою.

В UNIX-системах як інструмент автоматизації виступає стандартна оболонка (shell) або її модифікації (bashell, kshell, cshell і т.д.), причому цей аспект операційної системи стандартизований в рамках POSIX (стандарт мобільних систем).

В операційній системі Windows справа йде складніше. На сьогоднішній день одного "ідеального" засоби автоматизації, що задовольняє відразу всім перерахованим вище вимогам, в Windows немає; в останніх версіях операційної системи підтримуються кілька стандартних інструментів автоматизації, які сильно відрізняються один від одного: оболонка командного рядка ***cmd.exe***, середовище виконання сценаріїв ***Windows Script Host*** і оболонка ***Microsoft PowerShell***. Тому адміністратору або користувачу Windows доводиться вибирати, яким саме підходом скористатися для вирішення певної задачі, а для цього бажано мати чітке уявлення про сильні та слабкі сторони даних засобів автоматизації. Тут ми коротко обговоримо переваги і недоліки кожного з них (теоретичні та практичні подробиці будуть розглянуті в наступних лекціях).

## Оболонка командного рядка command.com/cmd.exe

У всіх версіях операційної системи Windows підтримується інтерактивна оболонка командного рядка (***command shell***) і за замовчуванням встановлюється певний набір утиліт командного рядка (кількість і склад цих утиліт залежить від версії операційної системи). Взагалі, будь-яку операційну систему можна представити у вигляді сукупності ядра системи, яке має доступ до апаратури і оперує файлами і процесами, і оболонки (командного інтерпретатора) з утилітами, які дозволяють користувачеві отримати доступ до функціональності ядра операційної системи. Механізм роботи оболонок в різних системах однаковий: у відповідь на запрошення ("підказку", prompt), що видається знаходиться в очікуванні оболонкою, користувач вводить деяку команду (функціональність цієї команди може бути реалізована або самою оболонкою, або певною зовнішньою утилітою), оболонка виконує її, при необхідності виводячи на екран будь-яку інформацію, після чого знову виводить запрошення і чекає введення наступної команди.

З технічної точки зору оболонка являє собою порядковий інтерпретатор простої мови сентенціального (директивного) програмування, у якості операторів якого можуть використовуватися виконувані програми.

Поряд з інтерактивним режимом роботи оболонки, як правило, підтримують і пакетний режим, в якому система послідовно виконує команди, записані в текстовому файлі-сценарії. Оболонка Windows не є винятком, з точки зору програмування мова командних файлів Windows може бути охарактеризована наступним чином:

* реалізація сентенціальної (директивної) парадигми програмування;
* виконання в режимі порядкової інтерпретації;
* наявність керуючих конструкцій;
* підтримка декількох видів циклів (в тому числі спеціальних циклів для обробки текстових файлів);
* наявність оператора присвоювання (встановлення значення змінної);
* можливість використання зовнішніх програм (команд) операційної системи в якості операторів і обробки їх кодів повернення;
* наявність нетипізованих змінних, які декларуються першою згадкою (значення змінних можуть інтерпретуватися як числа і використовуватися у виразах цілочисельної арифметики).

Починаючи з версії Windows NT, оболонка командного рядка представляється інтерпретатором ***Cmd.exe***, який розширює можливості оболонки ***command.com*** операційної системи MS-DOS. У свою чергу функціональність командного інтерпретатора ***command.com*** була запозичена з операційної системи CP/M, оболонка якої представляла собою значно спрощений і урізаний варіант оболонки Unix-систем.

Таким чином, оболонка командного рядка MS-DOS спочатку поступалася UNIX-оболонкам по зручності роботи і розвиненості мови сценаріїв; в командній оболонці Windows (***cmd.exe***), незважаючи на всі зроблені поліпшення, не вдалося подолати дане відставання ні в режимі інтерактивної роботи (наприклад, в ***cmd.exe*** відсутня підтримка псевдонімів для довгих назв команд і не реалізовано автоматичне завершення команд при введенні їх з клавіатури), ні в синтаксисі або можливості мови командних файлів. Ситуація ускладнювалася тим, що Windows завжди програвала UNIX-системам в кількості і функціональні можливості стандартних (що не вимагають додаткової установки) утиліт командного рядка, а також в якості і повноті вбудованої довідкової системи по командам оболонки.

На практиці проблему відсутності потрібної функціональності у стандартних команд доводиться вирішувати або за допомогою утиліт пакета Windows Resource Kit для відповідної версії операційної системи, або шляхом пошуку підходящої утиліти сторонніх виробників. Крім того, в Windows можна користуватися POSIX-сумісними утилітами і оболонками за допомогою пакета Microsoft Services For Unix (SFU). Даний продукт розроблявся ще для Windows NT і спочатку не входив до складу операційної системи, його потрібно було купувати за окрему плату. Надалі пакет SFU став безкоштовним і навіть був включений до складу операційної системи Windows Server 2003 R2.

Отже, з огляду на все сказане вище, ми можемо зробити наступний висновок: оболонка командного рядка ***cmd.exe*** і командні файли - найбільш універсальні і прості у вивченні засоби автоматизації роботи в Windows, доступні у всіх версіях операційної системи, які, однак, істотно програють аналогічним інструментам в UNIX-системах і не забезпечують доступ до об'єктним моделям, підтримуваним операційною системою (COM, WMI, .NET).

## Підтримка мов сценаріїв. Сервер сценаріїв Windows Script Host

Наступним кроком у розвитку засобів і технологій автоматизації в операційній системі Windows стала поява сервера сценаріїв ***Windows Script Host (WSH)***. Цей інструмент розроблено для всіх версій Windows і дозволяє безпосередньо в операційній системі виконувати сценарії на повноцінних мовах сценаріїв (за замовчуванням, VBScript і JScript), які до цього були доступні тільки всередині HTML-сторінок і працювали в контексті безпеки веб-браузера (в силу цього подібні сценарії, наприклад, могли не мати доступу до файлової системи локального комп'ютера).

У порівнянні з командними файлами інтерпретатора ***cmd.exe*** сценарії ***WSH*** мають кілька переваг.

По-перше, VBScript і JScript - це повноцінні алгоритмічні мови, мають вбудовані функції і методи для обробки символьних рядків, виконання математичних операцій, обробки виняткових ситуацій і т. д.; крім того, для написання сценаріїв ***WSH*** може використовуватися будь-яка інша мова сценаріїв (наприклад, широко поширений в UNIX-системах Perl), для якого встановлено відповідний модуль підтримки.

По-друге, ***WSH*** підтримує кілька власних об'єктів, властивості і методи яких дозволяють вирішувати деякі часто виникають повсякденні завдання адміністратора операційної системи: робота з мережевими ресурсами, змінними середовища, системним реєстром, ярликами і спеціальними папками Windows, запуск і управління роботою інших додатків.

По-третє, з сценаріїв ***WSH*** можна звертатися до служб будь-яких додатків-серверів автоматизації (наприклад, програм з пакету Microsoft Office), які реєструють в операційній системі свої об'єкти.

Нарешті, сценарії ***WSH*** дозволяють працювати з об'єктами інформаційної моделі Windows Management Instrumentation (WMI), що забезпечує програмний інтерфейс управління всіма компонентами операційної моделі, а також з об'єктами служби каталогів Active Directory Service Interface (ADSI) (об'єктні моделі WMI і ADSI будуть обговорюватися докладніше в наступних лекціях).

Слід також зазначити, що технологія ***WSH*** підтримується в Windows вже достатньо давно, в Інтернеті (в тому числі на сайті Microsoft) можна знайти безліч готових сценаріїв, які виконують ту чи іншу операцію і при певних навичках і знаннях швидко "підігнати" ці сценарії під свої конкретні завдання.

Поговоримо тепер про слабкі місця ***WSH***. Перш за все, сам по собі ***WSH*** - це тільки середовище виконання сценаріїв, а не оболонка; ***WSH*** не інтегрована з командним рядком, тобто відсутній режим, в якому можна було вводити команди з клавіатури і відразу бачити результат їх виконання.

Великим мінусом є те, що в операційній системі за замовчуванням немає повноцінної докладної довідкової інформації по об'єктах ***WSH*** і мов VBScript / JScript (документацію доводиться шукати в Інтернеті на сайті Microsoft). Іншими словами, якщо ви, наприклад, не пам'ятаєте синтаксис певної команди VBScript/JScript або точну назву властивості об'єкта ***WSH***, під рукою у вас немає роздрукованої документації, а комп'ютер не має виходу в Інтернет, то написати коректний сценарій вам просто не вдасться (в даному аспекті командні файли більш універсальні, так як практично у всіх команд є принаймні вбудований опис використовуваних ними ключів, а в операційній системі є довідковий файл з інформацією про всіх стандартних командах).

Нарешті, сценарії ***WSH*** представляють собою достатньо серйозну потенційну загрозу з точки зору безпеки, відома велика кількість вірусів, що використовують ***WSH*** для виконання деструктивних дій.

Таким чином, можна дати таку загальну оцінку: сценарії ***WSH*** - це універсальний інструмент, який в будь-якої версії операційної системи Windows дозволяє вирішувати завдання автоматизації практично будь-якого ступеня складності, але вимагає при цьому великої роботи по вивченню самих мов сценаріїв і ряду суміжних технологій управління операційною системою (WMI, ADSI і т.п.).

**Командна оболонка Microsoft PowerShell**

Отже, до початку XXI століття ситуацію із засобами автоматизації роботи в Windows не можна було назвати зовсім хорошою. З одного боку функціональності і гнучкості мови оболонки ***cmd.exe*** було явно недостатньо, а з іншого боку сценарії ***WSH***, що працюють з об'єктними моделями ADSI і WMI, виявилися занадто складними для користувачів середнього рівня і початківців адміністраторів.

У 2000 році було розпочато розробку нової оболонки для доступу до об'єктів WMI з командного рядка (WMI Command-line, WMIC). Цей продукт виявився не особливо вдалим, так як в ньому акцент був зроблений на функціональні особливості WMI, а не на зручність роботи користувача. Почавши допрацьовувати WMIC, фахівці Microsoft зрозуміли, що можна реалізувати оболонку, що не обмежувалася б тільки роботою з об'єктами WMI, а також надавала б доступ до будь-яких класів платформи .NET Framework, забезпечуючи тим самим можливість користуватися з командного рядка усіма потужними функціональними можливостями даного середовища.

Перед розробниками нової оболонки, що отримала назву ***Windows PowerShell*** (попередня назва - Monad), стояли такі основні цілі та завдання, які були успішно вирішені:

* застосування командного рядка в якості основного інтерфейсу адміністрування;
* реалізація моделі ObjectFlow (елементом обміну інформації є об'єкт);
* переробка існуючих команд, утиліт і оболонки;
* інтеграція командного рядка, об'єктів COM, WMI і .NET;
* робота з довільними джерелами даних в командному рядку за принципом файлової системи.

Взагалі, найважливіша ідея, закладена в ***PowerShell***, полягає в тому, що в командному рядку виведення результатів команди є не текст (в сенсі послідовності байтів), а об'єкт (дані разом з властивими їм методами). В силу цього працювати в ***PowerShell*** стає простіше, ніж в традиційних оболонках, так як не потрібно виконувати ніяких маніпуляцій по виділенню потрібної інформації з символьного потоку.

Крім того, розробники намагалися зібрати в ***PowerShell*** всі кращі аспекти інших оболонок командного рядка з різних операційних систем. За їх словами, сильний вплив на ***PowerShell*** надали такі продукти:

* BASH, KSH (конвейєризація або композиція команд);
* AS400/VMS (стандартні назви команд, що прискорюють вивчення);
* TCL/WSH (підтримка вбудовуваність і декількох мов);
* PERL, PYTHON (виразність і стиль).

Відзначимо, що ***PowerShell*** одночасно є і оболонкою командного рядка (користувач працює в інтерактивному режимі) і середовищем виконання сценаріїв, які пишуться на спеціальній мові ***PowerShell***.

Інтерактивний сеанс в ***PowerShell*** схожий на роботу в оболонці UNIX-систем: всі команди в ***PowerShell*** мають детальну вбудовану довідку (для більшості команд наводяться приклади їх використання), підтримується функція автоматичного завершення назв команд і їх параметрів при введенні з клавіатури, для багатьох команд є псевдоніми, аналогічні назвам UNIX-утиліт (ls, pwd, tee і т.д.).

Мова ***PowerShell*** нескладна для вивчення, писати на ньому сценарії, які звертаються до зовнішніх об'єктів, простіше, ніж на VBScript або JScript. Окрему увагу було приділено питанням безпеки при роботі зі сценаріями (наприклад, запустити сценарій можна тільки з зазначенням повного шляху до нього, а за замовчуванням запуск сценаріїв PowerShell в системі взагалі заборонений).

В цілому, оболонка ***PowerShell*** набагато зручніше і потужніше своїх попередників (***cmd.exe*** і ***WSH***), а основним недоліком, який стримує поширення нового інструменту, є той факт, що ***PowerShell*** працює не у всіх версіях операційної системи Windows. Оболонкою можна користуватися тільки на версіях не нижче Windows XP Service Pack 2 з встановленим пакетом .NET Framework 2.0.

**Оболонка командного рядка Windows. Інтерпретатор Cmd.exe**

В операційній системі Windows, як і в інших операційних системах, інтерактивні (набираються з клавіатури і відразу ж виконуються) команди виконуються за допомогою так званого командного інтерпретатора, інакше званого командним процесором або оболонкою командного рядка (***command shell***). Командний інтерпретатор або оболонка командного рядка - це програма, яка, перебуваючи в оперативній пам'яті, зчитує команди, які ви набираєте, і обробляє їх. У Windows 9x, як і в MS-DOS, командний інтерпретатор за замовчуванням був представлений виконуваним файлом ***command.com***. Починаючи з версії Windows NT, в операційній системі реалізований інтерпретатор команд ***cmd.exe***, що володіє набагато більш потужними можливостями.

**Запуск оболонки**

У Windows файл ***cmd.exe***, як і інші виконувані файли, відповідні зовнішнім командам операційної системи, знаходяться в каталозі ***%SystemRoot%\SYSTEM32*** (значенням змінної середовища % SystemRoot% є системний каталог Windows, зазвичай ***C:\ Windows*** або ***C:\WinNT***). Для запуску командного інтерпретатора (відкриття нового сеансу командного рядка) можна вибрати пункт Виконати ... (Run) в меню Пуск (Start), ввести ім'я файлу Cmd.exe і натиснути кнопку OK. В результаті відкриється нове вікно, в якому можна запускати команди і бачити результат їх роботи.

### Внутрішні і зовнішні команди. Структура команд

Деякі команди розпізнаються і виконуються безпосередньо самим командним інтерпретатором - такі команди називаються внутрішніми (наприклад, ***COPY*** або ***DIR***). Інші команди операційної системи являють собою окремі програми, розташовані за замовчуванням в тому ж каталозі, що і ***cmd.exe***, які Windows завантажує і виконує аналогічно іншим програмам. Такі команди називаються зовнішніми (наприклад, ***MORE*** або ***XCOPY***).

Розглянемо структуру самого командного рядка і принцип роботи з ним. Для того, щоб виконати команду, ви після запрошення командного рядка (наприклад, C:\>) вводите ім'я цієї команди (регістр не важливий), її параметри і ключі (якщо вони необхідні) та натиснути клавішу <Enter>. наприклад:

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

Ім'я команди тут - ***COPY***, параметри - ***C:\myfile.txt*** і ***A:\***, а ключем є ***/V***. Відзначимо, що в деяких командах ключі можуть починатися не з символу ***/***, а з символу ***-*** (мінус), наприклад, ***-V***.

Багато команд Windows мають велику кількість додаткових параметрів і ключів, запам'ятати які найчастіше буває важко. Більшість команд забезпечено вбудованою довідкою, в якій коротко описуються призначення і синтаксис даної команди. Отримати доступ до такої довідки можна шляхом введення команди з ключем ***/?***. Наприклад, якщо виконати команду ***ATTRIB /?***, то в вікні MS-DOS ми побачимо наступний текст:

Відображення і зміна атрибутів файлів.

***ATTRIB [+R|-R] [+A|-A] [+S|-S] [+H|-H] [[диск:] [шлях] ім'я файлу] [/S]***

***+ Установка атрибута.***

***- Зняття атрибута.***

***R Атрибут "Тільки читання".***

***A Атрибут "Архівний".***

***S Атрибут "Системний".***

***H Атрибут "Прихований".***

***/S Обробка файлів у всіх вкладених папках зазначеного шляху.***

Для деяких команд текст вбудованої довідки може бути достатньо великим і не уміщатися на одному екрані. У цьому випадку допомога можна виводити послідовно по одному екрану за допомогою команди ***MORE*** і символу конвейеризации |, наприклад:

***XCOPY /? | MORE***

У цьому випадку після заповнення чергового екрану виведення допомоги буде перериватися до натискання будь-якої клавіші. Крім того, використовуючи символи перенаправлення виведення ***>*** і ***>>***, можна текст, що виводиться на екран, направити в текстовий файл для подальшого перегляду. Наприклад, для виведення тексту довідки до команди ***XCOPY*** в текстовий файл xcopy.txt, використовується наступна команда:

***XCOPY /? > XCOPY.TXT***

Замість імені файлу можна вказувати позначення пристроїв комп'ютера. У Windows підтримуються наступні імена пристроїв: ***PRN*** (принтер), ***LPT1-LPT3*** (відповідні паралельні порти), ***AUX*** (пристрій, що приєднується до послідовного порту 1), ***COM1-COM3*** (відповідні послідовні порти), ***CON*** (термінал: при введенні це клавіатура, при виведенні - монітор), ***NUL*** (пустий пристрій, всі операції введення/виведення для нього ігноруються).

### Перенаправлення введення/виведення і конвеєризація (композиція) команд

Розглянемо більш докладно підтримувані в Windows UNIX-подібні концепції перепризначення пристроїв стандартного введення/виведення і конвеєрного виконання команд.

За допомогою перепризначення пристроїв введення/виведення одна програма може направити своє виведення на вхід іншого чи перехопити виведення іншої програми, використовуючи його в якості своїх вхідних даних. Таким чином, є можливість передавати інформацію від процесу до процесу при мінімальних програмних витратах. Практично це означає, що для програм, які використовують стандартні вхідні і вихідні пристрої, операційна система дозволяє:

* виводити повідомлення програм не на екран (стандартний вихідний потік), а в файл або на принтер (перенаправлення виведення);
* читати вхідні дані не з клавіатури (стандартний вхідний потік), а з заздалегідь підготовленого файлу (перенаправлення введення);
* передавати повідомлення, що виводяться однією програмою, в якості вхідних даних для іншої програми (конвеєризація або композиція команд).

З командного рядка ці можливості реалізуються в такий спосіб. Для того, щоб перенаправити текстові повідомлення, що виводяться будь-якої командою, в текстовий файл, потрібно використовувати конструкцію

***команда > ім’я\_файлу***

Якщо при цьому заданий для виведення файл вже існував, то він перезаписується (старий вміст втрачається), якщо не існував - створюється. Можна також не створювати файл заново, а дописувати інформацію, виведену командою, в кінець існуючого файлу. Для цього команда перенаправлення виведення повинна бути задана так:

***команда >> ім’я\_файлу***

За допомогою символу ***<*** можна прочитати вхідні дані для заданої команди не з клавіатури, а з певного (заздалегідь підготовленого) файлу:

***команда < ім'я\_файлу***

Наведемо кілька прикладів перенаправлення введення/виведення.

Виведення вбудованої довідки для команди ***COPY*** в файл ***copy.txt***:

***COPY /? > copy.txt***

Додавання тексту довідки для команди ***XCOPY*** в файл copy.txt:

***XCOPY /? >> copy.txt***

Введення нової дати з файлу ***date.txt*** (***DATE*** - це команда для перегляду і зміни системної дати):

***DATE < date.txt***

Якщо при виконанні певної команди виникає помилка, то повідомлення про це за замовчуванням виводиться на екран. У разі необхідності повідомлення про помилки (стандартний потік помилок) можна перенаправити в текстовий файл за допомогою конструкції

***команда 2> ім’я\_файлу***

В цьому випадку стандартне виведення буде проводитися в файл. Також є можливість інформаційні повідомлення та повідомлення про помилки виводити в один і той же файл. Робиться це в такий спосіб:

***команда > ім’я\_файлу 2>&1***

Наприклад, у наведеній нижче команді стандартний вихідний потік і стандартний потік помилок перенаправляються в файл copy.txt:

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

Нарешті, за допомогою конструкції

***команда1 | команда2***

можна використовувати повідомлення, що виводяться першою командою, в якості вхідних даних для другої команди (конвеєр команд).

Використовуючи механізми перенаправлення вводу / виводу і конвеєризації, можна з командного рядка посилати інформацію на різні пристрої і автоматизувати відповіді на запити, що видаються командами або програмами, які використовують стандартне введення. Для вирішення таких завдань підходить команда

***ECHO [повідомлення]***

яка виводить повідомлення на екран. Розглянемо приклади використання цієї команди.

Посилка символу прогону на принтер:

***ECHO ^L > PRN***

Видалення всіх файлів в поточному каталозі без попередження (автоматичний позитивну відповідь на запит про видалення):

***ECHO y | DEL \*.\****

З'єднання по телефону з командного рядка (модем пов'язаний з портом COM2):

***ECHO ATDT 1 (123) 555-1234 > COM2***

### Команди MORE і SORT

Однією з найбільш часто використовуються команд, для роботи з якою застосовується перенаправлення введення/виведення і конвеєризація, є ***MORE***. Ця команда зчитує стандартне введення з конвеєра або перенаправленого файлу і виводить інформацію частинами, розмір кожної з яких не більше розміру екрана. Використовується ***MORE*** зазвичай для перегляду довгих файлів. Можливі три варіанти синтаксису цієї команди:

***MORE [диск:] [шлях] ім'я\_файлу***

***MORE <[диск:] [шлях] ім'я\_файлу***

***імя\_команди | MORE***

Параметр ***[диск:] [шлях] ім'я\_файлу*** визначає розташування і ім'я файлу з кількістю переглядів на екрані даними. Параметр ***імя\_команди*** задає команду, виведення якої відображається на екрані (наприклад, ***DIR*** або команда ***TYPE***, що використовується для виведення вмісту текстового файлу на екран). Наведемо два приклади.

Для поекранного перегляду допомоги команди ***DIR*** використовується команда:

***DIR /? | MORE***

Для поекранного перегляду текстового файлу ***news.txt*** можливі наступні варіанти команд:

***MORE news.txt***

***MORE < news.txt***

***TYPE news.txt | MORE***

Іншою поширеною командою, що використовує перенаправлення введення/виведення і конвеєризацію, є ***SORT***. Ця команда працює як фільтр: вона зчитує символи в заданому стовпці, впорядковує їх у зростаючому або спадаючому порядку і виводить відсортовану інформацію в файл, на екран або інший пристрій. Можливі два варіанти синтаксису цієї команди:

***SORT [/R] [/+n] [[диск1:] [шлях1] файл1] [> [диск2:] [шлях2] файл2]***

або

***[Команда] | SORT [/R] [/+n] [> [диск2:] [путь2] файл2]***

У першому випадку параметр ***[диск1:] [шлях1] файл1*** визначає ім'я файлу, який потрібно відсортувати. У другому випадку будуть відсортовані вихідні дані зазначеної команди. Якщо параметри ***файл1*** або команда не задані, то ***SORT*** буде зчитувати дані з пристрою стандартного введення.

Параметр ***[диск2:] [шлях2] файл2*** задає файл, в який буде спрямовуватися сортовані висновок; якщо цей параметр не заданий, то висновок буде направлений на пристрій стандартного виведення.

За замовчуванням сортування виконується в порядку зростання. Ключ ***/R*** дозволяє змінити порядок сортування на зворотний (від Z до A і потім від 9 до 0). Наприклад, для поекранно перегляду відсортованого в зворотному порядку файлу price.txt, потрібно задати наступну команду:

***SORT /R < price.txt | MORE***

Ключ ***/+n*** задає сортування у файлі за символами n-го стовпця. Наприклад, ***/+10*** означає, що сортування повинна здійснюватися, починаючи з 10-ї позиції в кожному рядку. За замовчуванням файл сортується по на одну колонку.

### Умовне виконання і угруповання команд

У командному рядку Windows можна використовувати спеціальні символи, які дозволяють вводити кілька команд одночасно і управляти роботою команд в залежності від результатів їх виконання. За допомогою таких символів умовної обробки можна зміст невеликого пакетного файлу записати в одному рядку і виконати отриману складову команду.

Використовуючи символ амперсанда ***&***, можна розділити кілька утиліт в одній командному рядку, при цьому вони будуть виконуватися один за одним. Наприклад, якщо набрати команду ***DIR & PAUSE & COPY /?*** і натиснути клавішу ***<Enter>***, то спочатку на екран буде виведено вміст поточного каталогу, а після натискання будь-якої клавіші - вбудована довідка команди ***COPY***.

Символ ***^*** дозволяє використовувати командні символи як текст, тобто при цьому відбувається ігнорування значення спеціальних символів. Наприклад, якщо ввести в командному рядку

***ECHO Абв & COPY /?***

і натиснути клавішу ***<Enter>***, то станеться виконання поспіль двох команд: ***ECHO Абв і COPY /?*** (Команда ***ECHO*** виводить на екран символи, зазначені в командному рядку після неї). Якщо ж виконати команду

***ECHO Абв ^ & COPY /?***

то на екран буде виведено

***Абв & COPY /?***

У цьому випадку просто виконується одна команда ***ECHO*** з відповідними параметрами.

Умовна обробка команд в Windows здійснюється за допомогою символів ***&&*** і ***||*** наступним чином. Подвійний амперсанд ***&&*** запускає команду, яка стоїть за ним в командному рядку, тільки в тому випадку, якщо команда, що стоїть перед амперсандами була виконана успішно. Наприклад, якщо в кореневому каталозі диска ***C:*** є файл ***plan.txt***, то виконання рядки ***TYPE C:\plan.txt && DIR*** призведе до виведення на екран цього файлу і вмісту поточного каталогу. Якщо ж файл ***C:\plan.txt*** не існує, то команда ***DIR*** виконуватися не буде.

Два символи ***||*** здійснюють в командному рядку зворотну дію, тобто запускають команду, яка стоїть за цими символами, тільки в тому випадку, якщо команда, що йде перед ними, не була успішно виконана. Таким чином, якщо в попередньому прикладі файл ***C:\plan.txt*** буде відсутній, то в результаті виконання рядки ***TYPE C:\plan.txt || DIR*** на екран виведеться вміст поточного каталогу.

Відзначимо, що умовна обробка діє тільки на найближчу команду, тобто в рядку

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

команда ***COPY /?*** запуститься в будь-якому випадку, незалежно від результату виконання команди ***TYPE C:\plan.txt***.

Кілька утиліт можна згрупувати в командному рядку за допомогою дужок. Розглянемо, наприклад, два рядки:

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

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

У першій з них символ умовної обробки ***&&*** діє тільки на команду ***DIR***, в другій - одночасно на дві команди: ***DIR*** і ***COPY***.

# Лекція №2. «Робота з файловою системою Windows у командному рядку. Мова інтерпретатору Cmd.exe. Командні файли»

## Команди для роботи з файловою системою

Розглянемо деякі найбільш часто використовувані команди для роботи з файловою системою. Відзначимо спочатку кілька особливостей визначення шляхів до файлів в Windows. Нагадаємо, що файлова система логічно має деревоподібну структуру і імена файлів задаються в форматі ***[диск:] [шлях\] ім'я\_файлу***, тобто обов'язковим параметром є тільки ім'я файлу. При цьому, якщо шлях починається з символу "***\***", то маршрут обчислюється від кореневого каталогу, інакше - від поточного каталогу. Наприклад, ім'я ***C:123.txt*** задає файл 123.txt в поточному каталозі на диску C:, ***ім'я C:\123.txt*** - файл 123.txt в кореневому каталозі на диску C:, ім'я ***ABC\123.txt*** - файл 123 .txt в підкаталозі ABC поточного каталогу.

Існують особливі позначення для поточного каталогу і батьківського каталогів. Поточний каталог позначається символом ***.*** (точка), його батьківський каталог - символами ***..*** (дві точки). Наприклад, якщо поточним каталогом є ***C:\WINDOWS***, то шлях до файлу ***autoexec.bat*** в кореневому каталозі диска ***C:*** може бути записаний у вигляді ***..\autoexec.bat***.

В іменах файлів (але не дисків або каталогів) можна застосовувати так звані групові символи або шаблони: ***?*** (знак питання) і ***\**** (зірочка). Символ ***\**** в імені файлу означає будь-яку кількість будь-яких допустимих символів, символ ***?*** - один довільний символ або його відсутність. Скажімо, під шаблон ***text??1.txt*** підходять, наприклад, імена ***text121.txt*** і ***text911.txt***, під шаблон ***text\*.txt*** - імена ***text.txt***, ***textab12.txt***, а під шаблон ***text.\**** - всі файли з ім'ям ***text*** і довільним розширенням.

Для того, щоб використовувати довгі імена файлів при роботі з командним рядком, їх потрібно укладати в подвійні лапки. Наприклад, щоб запустити файл з ім'ям ***'Мій додаток.exe'*** з каталогу ***'Мої документи'***, потрібно в командному рядку набрати ***"C:\Мої документи\Мій додаток.exe"*** і натиснути клавішу ***<Enter>***.

Перейдемо тепер безпосередньо до командам для роботи з файловою системою.

### Команда CD

Поточний каталог можна змінити за допомогою команди

***CD [диск:] [шлях\]***

Шлях до необхідного каталогу вказується з урахуванням наведених вище зауважень. Наприклад, команда ***CD \*** виконує перехід в кореневий каталог поточного диска. Якщо запустити команду ***CD*** без параметрів, то на екран будуть виведені імена поточного диска і каталогу.

### Команда COPY

Однією з найбільш часто повторюваних завдань при роботі на комп'ютері є копіювання та переміщення файлів з одного місця в інше. Для копіювання одного або декількох файлів використовується команда ***COPY***.

Синтаксис цієї команди:

***COPY [/A | /B] джерело [/A | /B] [+ джерело [/A | /B] [+ ...]]   [Результат [/A | /B]] [/V] [/Y | /-Y]***

Короткий опис параметрів і ключів команди ***COPY*** приведено в табл. 2.1.

Табл. 2.1. Параметри та ключі команди ***COPY***

|  |  |
| --- | --- |
| **Ключ** | **Опис** |
| ***/A*** | Файл є текстовим файлом ASCII, тобто кінець файлу позначається символом з кодом ASCII 26 (***<Ctrl> + <Z>***) |
| ***/B*** | Файл є двійковим. Цей ключ вказує на те, що інтерпретатор команд повинен при копіюванні зчитувати з джерела число байт, заданий розміром в каталозі копійованого файлу |
| ***результат*** | Каталог для розміщення результату копіювання та / або ім'я створюваного файлу |
| ***/V*** | Перевірка правильності копіювання шляхом порівняння файлів після копіювання |
| ***/Y*** | Відключення режиму запиту підтвердження на заміну файлів |
| ***/-Y*** | Включення режиму запиту підтвердження на заміну файлів |

Наведемо приклади використання команди ***COPY***.

1. Копіювання файлу abc.txt з поточного каталогу в каталог ***D:\PROGRAM*** під тим же ім'ям:

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

1. Копіювання файлу ***abc.txt*** з поточного каталогу в каталог ***D:\PROGRAM*** під новим ім'ям ***def.txt***:

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

1. Копіювання всіх файлів з розширенням ***txt*** з диска ***A:*** у каталог ***'Мої документи'*** на диску ***C:***

***COPY A:\\*.txt "C:\Мої документи"***

Якщо не поставити в команді цільовий файл, то команда ***COPY*** створить копію файлу-джерела з тим же ім'ям, датою і часом створення, що і вихідний файл, і помістить нову копію в поточний каталог на поточному диску. Наприклад, для того, щоб скопіювати всі файли з кореневого каталогу диска ***A:*** у поточний каталог, досить виконати таку коротку команду:

***COPY A: \ \*.\****

Як джерело або результату при копіюванні можна вказувати імена не тільки файлів, але і пристроїв комп'ютера. Наприклад, для того, щоб роздрукувати файл ***abc.txt*** на принтері, можна скористатися командою копіювання цього файлу на пристрій ***PRN***:

***COPY abc.txt PRN***

Інший цікавий приклад: створимо новий текстовий файл і запишемо в нього інформацію без використання текстового редактора. Для цього достатньо ввести команду ***COPY CON my.txt***, яка буде копіювати те, що ви набираєте на клавіатурі, в файл ***my.txt*** (якщо цей файл існував, то він буде перезаписано, інакше - створиться). Для завершення введення необхідно ввести символ кінця файлу, тобто натиснути клавіші ***<Ctrl>+<Z>***.

Команда ***COPY*** може також об'єднувати (склеювати) кілька файлів в один. Для цього необхідно вказати єдиний результуючий файл і кілька вихідних. Це досягається шляхом використання групових знаків (***?*** і ***\****) або формату ***файл1+файл2+файл3***. Наприклад, для об'єднання файлів ***1.txt*** і ***2.txt*** в файл ***3.txt*** можна задати наступну команду:

***COPY 1.txt+2.txt 3.txt***

Об'єднання всіх файлів з розширенням ***dat*** з поточного каталогу в один файл ***all.dat*** може бути вироблено так:

***COPY /B \*.dat all.dat***

Ключ ***/B*** тут використовується для запобігання усічення файлів, що з'єднуються, так як при комбінуванні файлів команда ***COPY*** за замовчуванням вважає файли текстовими.

Якщо ім'я цільового файлу збігається з ім'ям одного з файлів, що копіюються, (крім першого) то початковий вміст цільового файлу втрачається. Якщо ім'я цільового файлу опущено, то в його якості використовується перший файл зі списку. Наприклад, команда ***COPY 1.txt+2.txt*** додасть до вмісту файлу ***1.txt*** вміст файлу ***2.txt***. Командою ***COPY*** можна скористатися і для присвоєння якому-небудь файлу поточної дати і часу без модифікації його вмісту. Для цього потрібно ввести команду типу

***COPY /B 1.txt+,,***

Тут коми вказують на пропуск параметра приймача, що і призводить до необхідного результату.

Команда ***COPY*** має і свої недоліки. Наприклад, з її допомогою можна копіювати приховані і системні файли, файли нульової довжини, файли з підкаталогів. Крім того, якщо при копіюванні групи файлів ***COPY*** зустріне файл, який в даний момент не можна скопіювати (наприклад, він зайнятий іншим додатком), то процес копіювання повністю перерветься, і інші файли не будуть скопійовані.

### Команда XCOPY

Зазначені при описі команди ***COPY*** проблеми можна вирішити за допомогою команди ***XCOPY***, яка надає набагато більше можливостей при копіюванні. Необхідно відзначити, правда, що ***XCOPY*** може працювати тільки з файлами і каталогами, але не з пристроями.

Синтаксис цієї команди:

***XCOPY джерело [результат] [ключі]***

Команда ***XCOPY*** має безліч ключів, ми торкнемося лише деяких з них.

1. Ключ ***/D [:[дата]]*** дозволяє копіювати тільки файли, змінені не раніше зазначеної дати. Якщо параметр дата не вказана, то копіювання буде проводитися тільки якщо джерело новіше результату. Наприклад, команда

***XCOPY "C:\Мої документи\\*.\*" "D:\BACKUP\Мої документи" /D***

скопіює в каталог ***'D:\BACKUP\Мої документи'*** тільки ті файли з каталогу ***'C:\Мої документи'***, які були змінені з часу останнього подібного копіювання або яких взагалі не було в ***'D:\BACKUP\Мої документи'***.

1. Ключ ***/S*** дозволяє копіювати всі непусті підкаталоги в каталозі-джерелі.
2. За допомогою ж ключа ***/E*** можна копіювати взагалі все підкаталоги, включаючи і порожні.
3. Якщо вказано ключ ***/C***, то копіювання триватиме навіть в разі виникнення помилок. Це буває дуже корисним при операціях копіювання, вироблених над групами файлів, наприклад, при резервному копіюванні даних.
4. Ключ ***/I*** важливий для випадку, коли копіюються кілька файлів, а файл призначення відсутній. При завданні цього ключа команда ***XCOPY*** вважає, що файл призначення повинен бути каталогом. Наприклад, якщо задати ключ ***/I*** в команді копіювання всіх файлів з розширенням ***txt*** з поточного каталогу в неіснуючий ще підкаталог ***TEXT***,

***XCOPY \*.txt TEXT /I***

то підкаталог ***TEXT*** буде створено без додаткових запитів.

1. Ключі ***/Q***, ***/F*** і ***/L*** відповідають за режим відображення при копіюванні. При завданні ключа ***/Q*** імена файлів при копіюванні не відображаються, ключа ***/F*** - відображаються повні шляхи джерела і результату. Ключ ***/L*** позначає, що відображаються тільки файли, які повинні бути скопійовані (при цьому саме копіювання не проводиться).
2. За допомогою ключа ***/H*** можна копіювати приховані і системні файли, а за допомогою ключа ***/R*** - замінювати файли з атрибутом "Тільки для читання". Наприклад, для копіювання всіх файлів з кореневого каталогу диска ***C:*** (включаючи системні та приховані) в каталог ***SYS*** на диску ***D:***, потрібно ввести наступну команду:

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

1. Ключ ***/T*** дозволяє застосовувати ***XCOPY*** для копіювання тільки структури каталогів джерела, без дублювання файлів, що знаходяться в цих каталогах, причому порожні каталоги і підкаталоги не включаються. Для того, щоб все ж включити порожні каталоги і підкаталоги, потрібно використовувати комбінацію ключів ***/T /E***.

9. Використовуючи ***XCOPY*** можна при копіюванні оновлювати тільки вже існуючі файли (нові файли при цьому не записуються). Для цього застосовується ключ ***/U***. Наприклад, якщо в каталозі ***C:​​\2*** знаходилися файли ***a.txt*** і ***b.txt***, а в каталозі ***C:​\1*** - файли ***a.txt***, ***b.txt***, ***c.txt*** і ***d.txt***, то після виконання команди

***XCOPY C:\1 C:\2 /U***

в каталозі ***C:​\2*** як і раніше залишаться лише два файли ***a.txt*** і ***b.txt***, вміст яких буде замінено вмістом відповідних файлів з каталогу ***C:\1***. Якщо за допомогою ***XCOPY*** копіювався файл з атрибутом "Тільки для читання", то за замовчуванням у файлу-копії цей атрибут зніметься. Для того, щоб копіювати не тільки дані, але і повністю атрибути файлу, необхідно використовувати ключ ***/K***.

1. Ключі ***/Y*** і ***/-Y*** визначають, чи потрібно запитувати підтвердження перед заміною файлів при копіюванні. ***/Y*** - придушення запиту підтвердження на перезапис існуючого цільового файлу, ***/-Y*** - запит підтвердження на перезапис існуючого цільового файлу.

### Команда DIR

Ще однією дуже корисною командою є ***DIR [диск:] [шлях] [ім'я\_файлу] [ключі]***, яка використовується для виведення інформації про вміст дисків і каталогів. Параметр ***[диск:] [шлях]*** задає диск і каталог, вміст якого потрібно вивести на екран. Параметр ***[ім’я\_файлу]*** задає файл або групу файлів, які потрібно включити в список. Наприклад, команда

***DIR C:\ \*.bat***

виведе на екран всі файли з розширенням ***bat*** в кореневому каталозі диска ***C:***. Якщо задати цю команду без параметрів, то виводиться мітка диска і його серійний номер, імена (в короткому і довгому варіантах) файлів і підкаталогів, що знаходяться в поточному каталозі, а також дата і час їх останньої модифікації. Після цього виводиться число файлів в каталозі, загальний обсяг (у байтах), яку він обіймав файлами, і обсяг вільного простору на диску. наприклад:

***Том в пристрої C має мітку PHYS1\_PART2***

***Серійний номер тому: 366D-6107***

***Вміст папки C: \ aditor***

***. <ПАПКА> 25.01.00 17:15.***

***.. <ПАПКА> 25.01.00 17:15 ..***

***TEMPLT02 DAT 227 07.08.98 1:00 templt02.dat***

***UNINST1 000 1 093 02.03.99 8:36 UNINST1.000***

***HILITE DAT 1 082 18.09.98 18:55 hilite.dat***

***……………………………………………………………………………………………………………………………………………………………***

***README TXT 3 974 25.01.00 17:26 readme.txt***

***ADITOR HLP 24 594 08.10.98 23:12 aditor.hlp***

***Текст ~ 1 TXT 0 11.03.01 9:02 Текстовий файл.txt***

***11 файлів 533 647 байт***

***2 папок 143 261 696 байт вільно***

За допомогою ключів команди ***DIR*** можна задати різні режими розташування, фільтрації і сортування. Наприклад, при використанні ключа ***/W*** перелік файлів виводиться в широкому форматі з максимально можливим числом імен файлів або каталогів на кожному рядку. наприклад:

***Том в пристрої C має мітку PHYS1\_PART2***

***Серійний номер тому: 366D-6107***

***Вміст папки C: \ aditor***

***[.] [..] TEMPLT02.DAT UNINST1.000 HILITE.DAT***

***TEMPLT01.DAT UNINST0.000 TTABLE.DAT ADITOR.EXE README.TXT***

***ADITOR.HLP текст ~ 1.TXT***

***11 файлів 533 647 байт***

***2 папок 143 257 600 байт вільно***

За допомогою ключа ***/A [[:]атрибути]*** можна вивести імена тільки тих каталогів і файлів, які мають задані атрибути (***R*** - "Тільки читання", ***A*** - "Архівний", ***S*** - "Системний", ***H*** - "Прихований", префікс "***-***" має значення НЕ). Якщо ключ ***/A*** використовується більш ніж з одним значенням атрибута, будуть виведені імена тільки тих файлів, у яких всі атрибути співпадають з заданими. Наприклад, для виведення імен всіх файлів в кореневому каталозі диска ***C:***, які одночасно є прихованими і системними, можна задати команду

***DIR C:\ /A:HS***

а для виведення всіх файлів, крім прихованих - команду

***DIR C:\ /A:-H***

Відзначимо тут, що атрибуту каталогу відповідає буква D, тобто для того, щоб, наприклад, вивести список всіх каталогів диска C :, потрібно задати команду

***DIR C:\ /A:D***

Ключ ***/O [[:]сортування]*** задає порядок сортування вмісту каталогу при виведенні його командою ***DIR***. Якщо цей ключ опущений, ***DIR*** друкує імена файлів і каталогів в тому порядку, в якому вони містяться в каталозі. Якщо ключ ***/O*** заданий, а параметр сортування не вказано, то ***DIR*** виводить імена в алфавітному порядку. У параметрі сортування можна використовувати такі значення: ***N*** - по імені (алфавітна), ***S*** - за розміром (починаючи з менших), ***E*** - за розширенням (алфавітна), ***D*** - за датою (починаючи з більш старих), ***A*** - за датою завантаження (починаючи з більш старих), ***G*** - почати список з каталогів. Префікс "***-***" означає зворотний порядок. Якщо задається більш одного значення порядку сортування, файли сортуються за першим критерієм, потім за другим і т.д.

Ключ ***/S*** означає виведення списку файлів із заданого каталогу і його підкаталогів.

Ключ ***/B*** перераховує тільки назви каталогів і імена файлів (в довгому форматі) по одному на рядок, включаючи розширення. При цьому виводиться тільки основна інформація, без підсумкової. наприклад:

***templt02.dat***

***UNINST1.000***

***hilite.dat***

***templt01.dat***

***UNINST0.000***

***ttable.dat***

***aditor.exe***

***readme.txt***

***aditor.hlp***

***текстовий файл.txt***

### Команди MKDIR і RMDIR

Для створення нового каталогу і видалення вже існуючого порожнього каталогу використовуються команди ***MKDIR [диск:] шлях*** і ***RMDIR [диск:] шлях [ключі]*** відповідно (або їх короткі аналоги ***MD*** і ***RD***). Наприклад:

***MKDIR "C:\Приклади"***

***RMDIR "C:\Приклади"***

Команда ***MKDIR*** не може бути виконана, якщо каталог або файл із заданим ім'ям вже існує. Команда ***RMDIR*** нічого очікувати виконано, якщо видаляється каталог не порожній.

### Команда DEL

Видалити один або кілька файлів можна за допомогою команди

***DEL [диск:] [шлях] ім'я\_файлу [ключі]***

Щоб видалити кілька файлів використовуються групові знаки ***?*** і ***\****. Ключ ***/S*** дозволяє видалити зазначені файли з усіх підкаталогів, ключ ***/F*** - примусово видворити файли, доступні тільки для читання, ключ ***/A [[:]атрибути]*** - вибирати файли для видалення по атрибутам (аналогічно ключу ***/A [[:]атрибути]*** в команді ***DIR***).

### Команда REN

Перейменувати файли і каталоги можна за допомогою команди ***RENAME*** (***REN***). Синтаксис цієї команди має наступний вигляд:

***REN [диск:] [шлях] [каталог1 | файл1] [каталог2 | файл2]***

Тут параметр ***каталог1 | файл1*** визначає назву каталогу/файлу, яке потрібно змінити, а ***каталог2 | файл2*** задає нову назву каталогу/файлу. У будь-якому параметрі команди ***REN*** можна використовувати групові символи ***?*** і ***\****. При цьому представлені шаблонами символи в параметрі ***файл2*** будуть ідентичні відповідним символам в параметрі ***файл1***. Наприклад, щоб змінити у всіх файлів з розширенням ***txt*** в поточній директорії розширення на ***doc***, потрібно ввести таку команду:

***REN \*.txt \*.doc***

Якщо файл з ім'ям ***файл2*** вже існує, то команда ***REN*** припинить виконання, і станеться висновок повідомлення, що файл вже існує або зайнятий. Крім того, в команді ***REN*** можна вказати інший диск або каталог для створення результуючих каталогу і файлу. Для цієї мети потрібно використовувати команду ***MOVE***, призначену для перейменування і переміщення файлів і каталогів.

### Команда MOVE

Синтаксис команди для переміщення одного або більше файлів має вигляд:

***MOVE [/Y | /-Y] [диск:] [шлях] імя\_файла1 [...] результуючий\_файл***

Синтаксис команди для перейменування папки має вигляд:

***MOVE [/Y | /-Y] [диск:] [шлях] каталог1 каталог2***

Тут параметр ***результуючий\_файл*** задає нове розміщення файлу і може включати ім'я диска, двокрапку, ім'я каталогу, або їх поєднання. Якщо переміщується тільки один файл, допускається вказати його нове ім'я. Це дозволяє відразу перемістити і перейменувати файл. Наприклад,

***MOVE "C:\Мої документи\список.txt" D:\list.txt***

Якщо вказано ключ ***/-Y***, то при створенні каталогів і заміні файлів буде видаватися запит на підтвердження. Ключ ***/Y*** скасовує видачу такого запиту.

## Мова інтерпретатора Cmd.exe. Командні файли

Мова оболонки командного рядка (***shell language***) в Windows реалізований у вигляді командних (або пакетних) файлів. Командний файл в Windows - це звичайний текстовий файл з розширенням bat або cmd, в якому записані допустимі команди операційної системи (як зовнішні, так і внутрішні), а також деякі додаткові інструкції та ключові слова, які надають командним файлам деяку схожість з алгоритмічними мовами програмування. Наприклад, якщо записати в файл deltmp.bat наступні команди:

***C:\***

***CD %TEMP%***

***DEL /F \*.tmp***

і запустити його на виконання (аналогічно виконуваних файлів з розширенням ***com*** або ***exe***), то ми видалимо всі файли в тимчасовій директорії Windows. Таким чином, виконання командного файлу призводить до того ж результату, що і послідовне введення записаних в ньому команд. При цьому не проводиться ніякої попередньої компіляції або перевірки синтаксису коду; якщо зустрічається рядок з помилковою командою, то вона ігнорується. Очевидно, що якщо вам доводиться часто виконувати одні і ті ж дії, то використання командних файлів може заощадити багато часу.

### Виведення повідомлень і дублювання команд

За замовчуванням команди пакетного файлу перед виконанням виводяться на екран, що виглядає не дуже естетично. За допомогою команди ***ECHO OFF*** можна відключити дублювання команд, що йдуть після неї (сама команда ***ECHO OFF*** при цьому все ж дублюється). Наприклад,

***REM Наступні дві команди будуть дублюватися на екрані ...***

***DIR C:\***

***ECHO OFF***

***REM А решта вже не будуть***

***DIR D:\***

Для відновлення режиму дублювання використовується команда ***ECHO ON***. Крім цього, можна відключити дублювання будь-якої окремої рядки в командному файлі, написавши на початку цього рядка символ ***@***, наприклад:

***ECHO ON***

***REM Команда DIR C: \ дублюється на екрані***

***DIR C:\***

***REM А команда DIR D: \ - немає***

***@DIR D:\***

Таким чином, якщо поставити в самий початок файлу команду

***@ECHO OFF***

то це вирішить всі проблеми з дублюванням команд.

У пакетному файлі можна виводити на екран рядки з повідомленнями. Робиться це за допомогою команди

***ECHO повідомлення***

наприклад,

***@ECHO OFF***

***ECHO Привіт!***

Команда ***ECHO.*** (Точка повинна слідувати безпосередньо за словом "ECHO") виводить на екран порожній рядок. наприклад,

***@ECHO OFF***

***ECHO Привіт!***

***ECHO.***

***ECHO Бувай!***

Часто буває зручно для перегляду повідомлень, що виводяться з пакетного файлу, попередньо повністю очистити екран командою ***CLS***.

Використовуючи механізм перенаправлення введення/виведення (символи ***>*** і ***>>***), можна направити повідомлення, що виводяться командою ***ECHO***, в певний текстовий файл. наприклад:

***@ECHO OFF***

***ECHO Привіт! > hi.txt***

***ECHO Бувай! >> hi.txt***

За допомогою такого методу можна, скажімо, заповнювати файли-протоколи зі звітом про проведені дії. наприклад:

***@ECHO OFF***

***REM Спроба копіювання***

***XCOPY C:\PROGRAMS D:\PROGRAMS /s***

***REM Додавання повідомлення в файл report.txt в разі***

***REM вдалого завершення копіювання***

***IF NOT ERRORLEVEL 1 ECHO Успішне копіювання >> report.txt***

### Використання параметрів командного рядка

При запуску пакетних файлів в командному рядку можна вказувати довільне число параметрів, значення яких можна використовувати всередині файлу. Це дозволяє, наприклад, застосовувати один і той же командний файл для виконання команд з різними параметрами.

Для доступу з командного файлу до параметрів командного рядка застосовуються символи ***%0***, ***%1***, ..., ***%9*** або ***%\****. При цьому замість ***%0*** підставляється ім'я виконуваного пакетного файлу, замість ***%1***, ***%2***, ..., ***%9*** - значення перших дев'яти параметрів командного рядка відповідно, а замість ***%\**** - всі аргументи. Якщо в командному рядку при виклику пакетного файлу задано менше дев'яти параметрів, то "зайві" змінні з ***%1*** - ***%9*** заміщуються порожніми рядками. Розглянемо наступний приклад. Нехай є командний файл ***copier.bat*** такого змісту:

***@ECHO OFF***

***CLS***

***ECHO Файл %0 копіює каталог %1 в %2***

***XCOPY %1 %2 /S***

Якщо запустити його з командного рядка з двома параметрами, наприклад

***copier.bat C:\Programs D:\Backup***

то на екран виведеться повідомлення

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

і станеться копіювання каталогу ***C:\Programs*** з усіма його підкаталогами в ***D:\Backup***.

При необхідності можна використовувати більш дев'яти параметрів командного рядка. Це досягається за допомогою команди ***SHIFT***, яка змінює значення заміщаються параметрів з ***%0*** по ***%9***, копіюючи кожен параметр в попередній, тобто значення ***%1*** копіюється в ***%0***, значення ***%2*** – в ***%1*** і т.д. Замість параметру ***%9*** присвоюється значення параметра, наступного в командному рядку за старим значенням ***%9***. Якщо ж такий параметр не заданий, то нове значення ***%9*** - порожній рядок.

Розглянемо приклад. Нехай командний файл ***my.bat*** викликаний з командного рядка таким чином:

***my.bat p1 p2 p3***

Тоді ***%0=my.bat***, ***%1=p1***, ***%2=p2***, ***%3=p3***, параметри ***%4-%9*** є порожніми рядками. Після виконання команди ***SHIFT*** значення параметрів зміняться наступним чином: ***%0=p1***, ***%1=p2***, ***%2=p3***, параметри ***%3-%9*** - порожні рядки.

При включенні розширеної обробки команд SHIFT підтримує ключ ***/n***, що задає початок зсуву параметрів з номера ***n***, де ***n*** може бути числом від 0 до 9.

Наприклад, в наступній команді:

***SHIFT /2***

Параметр ***%2*** замінюється на ***%3***, ***%3*** на ***%4*** і т.д., а параметри ***%0*** і ***%1*** залишаються без змін.

Команда, зворотна ***SHIFT*** (зворотний зсув), відсутня. Після виконання ***SHIFT*** вже не можна відновити параметр (***%0***), який був першим перед зрушенням. Якщо в командному рядку задано більше десяти параметрів, то команду ***SHIFT*** можна використовувати кілька разів.

У командних файлах є деякі можливості синтаксичного аналізу замінних параметрів. Для параметра з номером ***n*** (***%n***) допустимі синтаксичні конструкції (оператори), представлені у табл. 2.2.

Табл. 2.2. Спеціальні змінні для синтаксичного аналізу командного рядка

|  |  |
| --- | --- |
| **Конструкція** | **Опис** |
| ***%~Fn*** | Змінна ***%n*** розширюється до повного імені файлу |
| ***%~Dn*** | Із змінної ***%n*** виділяється тільки ім'я диска |
| ***%~Pn*** | Із змінної ***%n*** виділяється тільки шлях до файлу |
| ***%~Nn*** | Із змінної ***%n*** виділяється тільки ім'я файлу |
| ***%~Xn*** | Із змінної ***%n*** виділяється розширення імені файлу |
| ***%~Sn*** | Значення операторів ***N*** та ***X*** для змінної ***%n*** змінюється так, що вони працюють з коротким ім'ям файлу |
| ***%~$PATH:n*** | Проводиться пошук по каталогам, заданим в змінного середовища PATH, і змінна% n замінюється на повне ім'я першого знайденого файлу |

Якщо змінна ***PATH*** не визначена або в результаті пошуку не знайдено жодного файлу, ця конструкція замінюється на порожній рядок. Природно, тут змінну ***PATH*** можна замінити на будь-яке інше допустиме значення.

Дані синтаксичні конструкції можна об'єднувати один з одним, наприклад:

***%~DPn*** - з змінної ***%n*** виділяється ім'я диска і шлях,

***%~NXn*** - з змінної ***%n*** виділяється ім'я файлу і розширення.

Розглянемо наступний приклад. Нехай ми знаходимося в каталозі ***C:​\TEXT*** і запускаємо пакетний файл з параметром ***Розповідь.doc*** (***%1=Розповідь.doc***). Тоді застосування операторів, описаних в таблиці вище, до параметру ***%1*** дасть наступні результати:

***%~F1=C:\TEXT\ Розповідь.doc***

***%~D1=C:***

***%~P1=\TEXT\***

***%~N1=Розповідь***

***%~X1=.doc***

***%~DP1=C:\TEXT\***

***%~NX1=Розповідь.doc***

### Робота зі змінними середовища

Усередині командних файлів можна працювати з так званими змінними середовища (або змінними оточення), кожна з яких зберігається в оперативній пам'яті, має своє унікальне ім'я, а її значенням є рядок. Стандартні змінні середовища автоматично не започатковано в процесі завантаження операційної системи. Такими змінними є, наприклад, ***WINDIR***, яка визначає розташування каталогу Windows, ***TEMP***, яка визначає шлях до каталогу для зберігання тимчасових файлів Windows або ***PATH***, в якій зберігається системний шлях (шлях пошуку), тобто список каталогів, в яких система повинна шукати виконувані файли або файли спільного доступу (наприклад, динамічні бібліотеки). Крім того, в командних файлах за допомогою команди ***SET*** можна оголошувати власні змінні середовища.

**Отримання значення змінної**

Для отримання значення певної змінної середовища потрібно ім'я цієї змінної укласти в символи%. наприклад:

***@ECHO OFF***

***CLS***

***REM Створення змінної MyVar***

***SET MyVar=Привіт***

***REM Зміна змінної***

***SET MyVar=%MyVar%!***

***ECHO Значення змінної MyVar: %MyVar%***

***REM Видалення змінної MyVar***

***SET MyVar=***

***ECHO Значення змінної WinDir: %WinDir%***

При запуску такого командного файлу на екран виведеться рядок

Значення змінної ***MyVar***: ***Привіт!***

Значення змінної ***WinDir***: ***C:\WINDOWS***

### **Перетворення** змінних як рядків

Зі змінними середовища в командних файлах можна виробляти деякі маніпуляції. По-перше, над ними можна робити операцію конкатенації (склеювання). Для цього потрібно в команді SET просто написати поруч значення з'єднуються змінних. наприклад,

***SET A=Раз***

***SET B=Два***

***SET C=%A%%B%***

Після виконання в файлі цих команд значенням змінної ***C*** буде рядок ***'раздва'***. Не слід для конкатенації використовувати знак ***+***, так як він буде сприйнятий просто як символ. Наприклад, після запуску файлу такого змісту

***SET A=Раз***

***SET B=Два***

***SET C=A+B***

***ECHO Змінна C=%C%***

***SET D=%A%+%B%***

***ECHO Змінна D=%D%***

***на екран виведуться два рядки:***

***Змінна C=A+B***

***Змінна D=Раз+Два***

По-друге, з змінної середовища можна виділяти підрядок за допомогою конструкції ***%ім'я\_змінної:~n1,n2%***, де число ***n1*** визначає зміщення (кількість пропускаються символів) від початку (якщо ***n1*** позитивно) або від кінця (якщо ***n1*** негативно) відповідної змінної середовища, а число ***n2*** - кількість виділених символів (якщо ***n2*** позитивно) або кількість останніх символів у змінній, які не ввійдуть в виділяється підрядок (якщо ***n2*** негативно). Якщо вказано тільки один негативний параметр ***-n***, то будуть вилучені останні ***n*** символів. Наприклад, якщо у змінній ***%DATE%*** зберігається рядок ***"21.09.2007"*** (символьне представлення поточної дати при певних регіональних нааштуваннях), то після виконання наступних команд

***SET dd1=%DATE:~0,2%***

***SET dd2=%DATE:~0,-8%***

***SET mm=%DATE:~-7,2%***

***SET yyyy=%DATE:~-4%***

нові змінні матимуть такі значення: ***%dd1%=21***, ***%dd2%=21***, ***%mm%=09***, ***%yyyy%=2007***.

По-третє, можна виконувати процедуру заміни підрядків за допомогою конструкції ***%ім'я\_змінної:s1=s2%*** (в результаті буде повернута рядок, в якій кожне входження підрядка ***s1*** у відповідній змінної середовища замінено на ***s2***). Наприклад, після виконання команд

***SET a=123456***

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

в змінній ***b*** буде зберігатися рядок ***"199456"***. Якщо параметр ***s2*** не вказано, то підрядок ***s1*** буде видалена з рядка, що виводиться, тобто після виконання команди

***SET a=123456***

***SET b=%a:23=%***

в змінній ***b*** буде зберігатися рядок ***"1456"***.

### Операції зі змінними як з числами

При включеній розширеній обробці команд (цей режим в Windows використовується за замовчанням) є можливість розглядати значення змінних середовища як числа і проводити з ними арифметичні обчислення. Для цього використовується команда ***SET*** з ключем ***/A***. Наведемо приклад пакетного файлу ***add.bat***, складаються два числа, заданих як параметри командного рядка, і виводить отриману суму на екран:

***@ECHO OFF***

***REM В змінної M буде зберігатися сума***

***SET /A M=%1+%2***

***ECHO Сума %1 і %2 дорівнює %M%***

***REM Вилучимо змінну M***

***SET M =***

### Локальні зміни змінних

Всі зміни, вироблені з допомогою команди ***SET*** над змінними середовища в командному файлі, зберігаються і після завершення роботи цього файлу, але діють тільки всередині поточного командного вікна. Також є можливість локалізувати зміни змінних середовища всередині пакетного файлу, тобто автоматично відновлювати значення всіх змінних в тому вигляді, в якому вони були до початку запуску цього файлу. Для цього використовуються дві команди: ***SETLOCAL*** і ***ENDLOCAL***. Команда ***SETLOCAL*** визначає початок області локальних установок змінних середовища. Іншими словами, зміни середовища, внесені після виконання ***SETLOCAL***, будуть локальними щодо поточного пакетного файлу. Кожна команда ***SETLOCAL*** повинна мати відповідну команду ***ENDLOCAL*** для відновлення колишніх значень змінних середовища. Зміни середовища, внесені після виконання команди ***ENDLOCAL***, вже не є локальними щодо поточного пакетного файлу; їх колишні значення не будуть відновлені після закінчення виконання цього файлу.

### Призупинення виконання командних файлів

Для того, щоб вручну перервати виконання запущеного bat-файлу, потрібно натиснути клавіші ***<Ctrl>+<C>*** або ***<Ctrl>+<Break>***. Однак часто буває необхідно програмно призупинити виконання командного файлу в певному рядку з видачею запиту на натискання будь-якої клавіші. Це робиться за допомогою команди ***PAUSE***. Перед запуском цієї команди корисно за допомогою команди ***ECHO*** інформувати користувача про дії, які він повинен зробити. наприклад:

***ECHO Вставте дискету в дисковод A: і натисніть будь-яку клавішу***

***PAUSE***

Команду ***PAUSE*** обов'язково потрібно використовувати при виконанні потенційно небезпечних дій (видалення файлів, форматування дисків і т.п.). Наприклад,

***ECHO Зараз будуть видалені всі файли в C:\Мої документи!***

***ECHO Для скасування натисніть Ctrl-C***

***PAUSE***

***DEL "C:\Мої документи\\*.\*"***

### Виклик зовнішніх командних файлів

З одного командного файлу можна викликати інший, просто вказавши його ім'я. наприклад:

***@ECHO OFF***

***CLS***

***REM Виведення списку log-файлів***

***DIR C:\\*.Log***

***REM Передача виконання файлу f.bat***

***f.bat***

***COPY A:\\*.\* C:\***

***PAUSE***

Однак в цьому випадку після виконання викликаного файлу управління в файл, викликаться, не передається, тобто в наведеному прикладі команда

***COPY A:\\*.\* C:\***

(І всі наступні за нею команди) ніколи не буде виконані.

Для того, щоб викликати зовнішній командний файл з наступним поверненням в початковий файл, потрібно використовувати спеціальну команду ***CALL файл***.

Наприклад:

***@ECHO OFF***

***CLS***

***REM Виведення списку log-файлів***

***DIR C:\\*.Log***

***REM Передача виконання файлу f.bat***

***CALL f.bat***

***COPY A:\\*.\* C:\***

***PAUSE***

У цьому випадку після завершення роботи файлу ***f.bat*** управління повернеться в первинний файл на рядок, наступну за командою ***CALL*** (в нашому прикладі це команда ***COPY A:\\*.\* C:\***).

# Лекція №3. «Основи програмування командних файлів Windows»

## Оператори переходу у командних файлах

Командний файл може містити мітки і команди ***GOTO*** переходу до цими мітками. Будь-який рядок, що починається з двокрапки, сприймається при обробці командного файлу як мітка. Ім'я мітки задається набором символів, що настають за двокрапкою до першого пробілу або кінця рядка. Наведемо приклад.

Нехай є командний файл такого змісту:

***@ECHO OFF***

***COPY %1 %2***

***GOTO Label1***

***ECHO Цей рядок ніколи не виконається***

***:Label1***

***REM Продовження виконання***

***DIR %2***

Після того, як в цьому файлі ми доходимо до команди

***GOTO Label1***

його виконання продовжується з рядка

***REM Продовження виконання***

У команді переходу всередині файлу ***GOTO*** можна задавати в якості мітки переходу рядок ***:EOF***, яка передає управління в кінець поточного пакетного файлу (це дозволяє легко вийти з пакетного файлу без визначення будь-яких позначок в самому його кінці).

Також для переходу до мітки всередині поточного командного файлу крім команди ***GOTO*** можна використовувати і розглянуту вище команду ***CALL***:

***CALL: мітка\_аргументу***

При виклику такої команди створюється новий контекст поточного пакетного файлу з заданими аргументами, і управління передається на інструкцію, розташовану відразу після мітки. Для виходу з такого пакетного файлу необхідно два рази досягти його кінця. Перший вихід повертає управління на інструкцію, розташовану відразу після рядка ***CALL***, а другий вихід завершує виконання пакетного файлу. Наприклад, якщо запустити з параметром ***"Копія-1"*** командний файл такого змісту:

***@ECHO OFF***

***ECHO% 1***

***CALL :2 Копія-2***

***:2***

***ECHO% 1***

то на екран виведуться три рядки:

***Копія-1***

***Копія-2***

***Копія-1***

Таким чином, подібне використання команди ***CALL*** дуже схоже на звичайний виклик підпрограм (процедур) в алгоритмічних мовах програмування.

## Оператори умови у командних файлах

За допомогою команди ***IF ... ELSE*** (ключове слово ***ELSE*** може бути відсутнім) в пакетних файлах можна виконувати обробку умов декількох типів. При цьому якщо задана після ***IF*** умова приймає істинне значення, система виконує наступну за умовою команду (або кілька команд, укладених в круглі дужки), в іншому випадку виконується команда (або кілька команд в дужках), наступні за ключовим словом ***ELSE***.

### Перевірка значення змінної

Перший тип умови використовується зазвичай для перевірки значення змінної. Для цього застосовуються два варіанти синтаксису команди ***IF***:

***IF [NOT] рядок1==рядок2 команда1 [ELSE команда2]***

(Квадратні дужки вказують на необов'язковість укладених в них параметрів) або

***IF [/I] [NOT] рядок1 оператор\_порівняння рядок2 команда***

Розглянемо спочатку перший варіант. Умова ***рядок1==рядок2*** (тут необхідно писати саме два знака рівності) вважається дійсним при точному збігу обох рядків. Параметр ***NOT*** вказує на те, що задана команда виконується лише в тому випадку, коли порівнювані рядки не збігаються.

Рядки можуть бути літеральними або являти собою значення змінних (наприклад, ***%1*** або ***%TEMP%***). Лапки для літеральних рядків не потрібні. наприклад,

***IF %1==%2 ECHO Параметри збігаються!***

***IF %1==Петро ECHO Привіт, Петро!***

Відзначимо, що при порівнянні рядків, заданих змінними, слід проявляти певну обережність. Справа в тому, що значення змінної може виявитися порожнім рядком, і тоді може виникнути ситуація, при якій виконання командного файлу аварійно завершиться. Наприклад, якщо ви не визначили за допомогою команди ***SET*** змінну ***MyVar***, а в файлі є умовний оператор типу

***IF %MyVar%==C:\ ECHO Ура!!!***

то в процесі виконання замість ***%MyVar%*** підставить порожній рядок і виникне синтаксична помилка. Така ж ситуація може виникнути, якщо один з порівнюваних рядків є значенням параметра командного рядка, так як цей параметр може бути не вказано при запуску командного файлу. Тому при порівнянні рядків потрібно приписувати до них на початку який-небудь символ, наприклад:

***IF -%MyVar%==-C:\ECHO Ура!!!***

За допомогою команд ***IF*** і ***SHIFT*** можна в циклі обробляти всі параметри командного рядка файлу, навіть не знаючи заздалегідь їх кількості. Наприклад, наступний командний файл (назвемо його ***pryklad.bat***) виводить на екран ім'я запускається файлу і всі параметри командного рядка:

***@ECHO OFF***

***ECHO Виконується файл: %0***

***ECHO.***

***ECHO Файл запущений з наступними параметрами ...***

***REM Початок циклу***

***:BegLoop***

***IF -%1==- GOTO ExitLoop***

***ECHO %1***

***REM Зрушення параметрів***

***SHIFT***

***REM Перехід на початок циклу***

***GOTO BegLoop***

***: ExitLoop***

***REM Вихід з циклу***

***ECHO.***

***ECHO Все.***

Якщо запустити ***primer.bat*** з чотирма параметрами:

***primer.bat А Б В Г***

то в результаті виконання на екран виведеться наступна інформація:

***Виконується файл: primer.bat***

***Файл запущений з наступними параметрами:***

***А***

***Б***

***В***

***Г***

***Все.***

Розглянемо тепер оператор ***IF*** в наступному вигляді:

***IF [/I] рядок1 оператор\_порівняння рядок2 команда***

Синтаксис і значення операторів порівняння представлені у табл. 3.1.

Табл. 3.1. Оператори порівняння

|  |  |
| --- | --- |
| **оператор** | **значення** |
| ***EQL*** | ***=*** |
| ***NEQ*** | ***≠*** |
| ***LSS*** | ***<*** |
| ***LEQ*** | ***≤*** |
| ***GTR*** | ***>*** |
| ***GEQ*** | ***≥*** |

Наведемо приклад використання операторів порівняння:

***@ECHO OFF***

***CLS***

***IF -%1 EQL -Василь ECHO Привіт, Василь!***

***IF -%1 NEQ -Василь ECHO Привіт, але ви не Василь!***

Ключ ***/I***, якщо він зазначений, задає порівняння текстових рядків без урахування регістру. Ключ ***/I*** можна також використовувати і в формі ***рядок1==рядок2*** команди ***IF***. Наприклад, умова

***IF /I DOS==dos ...***

буде істинною.

**Перевірка існування заданого файлу**

Другий спосіб використання команди ***IF*** - це перевірка існування заданого файлу. Синтаксис для цього випадку має вигляд:

***IF*** ***[NOT] EXIST файл команда1 [ELSE команда2]***

Умова вважається дійсною, якщо зазначений файл існує. Лапки для імені файлу не потрібні. Наведемо приклад командного файлу, в якому за допомогою такого варіанту команди ***IF*** перевіряється наявність файлу, вказаного в якості параметра командного рядка.

***@ECHO OFF***

***IF -%1==- GOTO NoFileSpecified***

***IF NOT EXIST %1 GOTO FileNotExist***

***REM Висновок повідомлення про знайдений файлі***

***ECHO Файл '%1' успішно знайдений.***

***GOTO :EOF***

***:NoFileSpecified***

***REM Файл запущений без параметрів***

***ECHO У командному рядку не вказано ім'я файлу.***

***GOTO :EOF***

***:FileNotExist***

***REM Параметр командного рядка заданий, але файл не знайдений***

***ECHO Файл '%1' не знайдений.***

### Перевірка наявності змінної середовища

Аналогічно файлів команда ***IF*** дозволяє перевірити наявність в системі певної змінної середовища:

***IF*** ***DEFINED змінна команда1 [ELSE команда2]***

Тут умова ***DEFINED*** застосовується подібно умові ***EXISTS*** наявності заданого файлу, але приймає в якості аргументу ім'я змінної середовища і повертає істинне значення, якщо ця змінна визначена. наприклад:

***@ECHO OFF***

***CLS***

***IF DEFINED MyVar GOTO: VarExists***

***ECHO Змінна MyVar не визначена***

***GOTO :EOF***

***: VarExists***

***ECHO Змінна MyVar визначена,***

***ECHO її значення дорівнює %MyVar%***

### Перевірка коду завершення попередньої команди

Ще один спосіб використання команди ***IF*** - це перевірка коду завершення (коду виходу) попередньої команди. Синтаксис для ***IF*** в цьому випадку має такий вигляд:

***IF*** ***[NOT] ERRORLEVEL число команда1 [ELSE команда2]***

Тут умова вважається дійсною, якщо остання запущена команда або програма завершилася з кодом повернення, що є рівним або перевищує вказане число.

Складемо, наприклад, командний файл, який би копіював файл ***my.txt*** на диск ***C:*** без виведення на екран повідомлень про копіювання, а в разі виникнення будь-якої помилки видавав попередження:

***@ECHO OFF***

***XCOPY my.txt C: \> NUL***

***REM Перевірка коду завершення копіювання***

***IF ERRORLEVEL 1 GOTO ErrOccurred***

***ECHO Копіювання виконано без помилок.***

***GOTO: EOF***

***: ErrOccurred***

***ECHO При виконанні команди XCOPY виникла помилка!***

В операторі ***IF ERRORLEVEL ...*** можна також застосовувати оператори порівняння чисел, наведені в таблиці вище:

***IF ERRORLEVEL LEQ ​​1 GOTO Case1***

Іноді зручнішим для роботи з кодами завершення програм може виявитися використання змінної ***%ERRORLEVEL%***. (Строкове представлення поточного значення коду помилки ***ERRORLEVEL***).

### Перевірка версії реалізації розширеної обробки команд

Нарешті, для визначення внутрішнього номера версії поточної реалізації розширеної обробки команд застосовується оператор ***IF*** в наступному вигляді:

***IF CMDEXTVERSION число команда1 [ELSE команда2]***

Тут умова ***CMDEXTVERSION*** застосовується подібно умові ***ERRORLEVEL***, але число порівнюється з вищезазначеним внутрішнім номером версії. Перша версія має номер ***1***. Номер версії буде збільшуватися на одиницю при кожному додаванні істотних можливостей розширеної обробки команд. Якщо розширена обробка команд відключена, умова ***CMDEXTVERSION*** ніколи не буває істинно.

## Організація циклів у командних файлах

У командних файлах для організації циклів використовуються кілька різновидів оператора ***FOR***, які забезпечують такі функції:

* виконання заданої команди для всіх елементів зазначеної множини;
* виконання заданої команди для всіх відповідних імен файлів;
* виконання заданої команди для всіх відповідних імен каталогів;
* виконання заданої команди для певного каталогу, а також всіх його підкаталогів;
* отримання послідовності чисел із заданими початком, кінцем і кроком збільшення;
* читання і обробка рядків з текстового файлу;
* обробка рядків виведення певної команди.

### Цикл FOR ... IN ... DO ...

Найпростіший варіант синтаксису команди ***FOR*** для командних файлів має такий вигляд:

***FOR %%змінна IN (безліч)***

***DO команда [параметри]***

**Увага!** Перед назвою змінної повинні стояти саме два знака відсотка (***%%***), а не один, як це було при використанні команди ***FOR*** безпосередньо з командного рядка.

Відразу наведемо приклад. Якщо в командному файлі задані рядки

***@ECHO OFF***

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

то в результаті його виконання на екрані буде надруковано наступне:

***Раз***

***Два***

***Три***

Параметр ***множина*** в команді ***FOR*** задає одну або більше текстових рядків, розділених комами, які ви хочете обробити за допомогою заданої команди. Дужки тут обов'язкові. Параметр ***команда [параметри]*** задає команду, виконувану для кожного елемента множини, при цьому вкладеність команд ***FOR*** на одному рядку не допускається. Якщо в рядку, що входить у множину, використовується кома, то значення цього рядка потрібно взяти в лапки. Наприклад, в результаті виконання файлу з командами

***@ECHO OFF***

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

***на екран буде виведено***

***Раз, Два***

***Три***

Параметр ***%%змінна*** дає підставляти змінну (лічильник циклу), причому тут можуть використовуватися тільки імена змінних, що складаються з однієї літери. При виконанні команда ***FOR*** замінює змінну, що підставляється текстом кожного рядка в заданій множині, поки команда, що стоїть після ключового слова ***DO***, що не обробить всі такі рядки.

Щоб уникнути плутанини з параметрами командного файлу ***%0-%9***, для змінних слід використовувати будь-які символи крім ***0-9***.

Параметр ***множина*** в команді ***FOR*** може також представляти одну або кілька груп файлів. Наприклад, щоб вивести в файл список всіх файлів з розширеннями ***txt*** і ***prn***, що знаходяться в каталозі ***C:​\TEXT***, без використання команди ***DIR***, можна використовувати командний файл такого змісту:

***@ECHO OFF***

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

При такому використанні команди ***FOR*** процес обробки триває, поки не оброблені всі файли (або групи файлів), зазначені у множині.

### Цикл **FOR** /D ... IN ... DO ...

Наступний варіант команди ***FOR*** реалізується за допомогою ключа ***/D***:

***FOR /D %%змінна IN (набір) DO команда [параметри]***

У разі, якщо набір містить символи узагальнення, то команда виконується для всіх відповідних імен каталогів, а не імен файлів. Скажімо, виконавши наступний командний файл:

***@ECHO OFF***

***CLS***

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

ми отримаємо список всіх каталогів на диску C :, наприклад:

***C:\Arc***

***C:\CYR***

***C:\MSCAN***

***C:\NC***

***C:\Program Files***

***C:\TEMP***

***C:\TeX***

***C:\WINNT***

**Цикл FOR /R ... IN ... DO ...**

За допомогою ключа ***/R*** можна задати рекурсію в команді ***FOR***:

***FOR /R [[диск:] шлях] %%змінна IN (набір)***

***DO команда [параметри]***

В цьому випадку задана команда виконується для каталогу ***[диск:] шлях***, а також для всіх підкаталогів цього шляху. Якщо після ключа ***/R*** не вказано ім'я каталогу, то виконання команди починається з поточного каталогу. Наприклад, для роздруківки всіх файлів з розширенням ***txt*** в поточному каталозі і всіх його підкаталогах можна використовувати наступний пакетний файл:

***@ECHO OFF***

***CLS***

***FOR /R %%f IN (\*.txt) DO PRINT %%f***

Якщо замість набору вказана тільки точка (***.***), то команда перевіряє всі підкаталоги поточного каталогу. Наприклад, якщо ми знаходимося в каталозі ***C:​\TEXT*** з двома підкаталогами ***BOOKS*** і ***ARTICLES***, то в результаті виконання файлу:

***@ECHO OFF***

***CLS***

***FOR /R %%f IN (.) DO ECHO %%f***

на екран виведуться три рядки:

***C:\TEXT\.***

***C:\TEXT\BOOKS\.***

***C:\TEXT\ARTICLES\.***

### Цикл FOR / L ... IN ... DO ...

Ключ ***/L*** дозволяє реалізувати за допомогою команди ***FOR*** арифметичний цикл, в цьому випадку синтаксис має наступний вигляд:

***FOR /L %%змінна IN (початок, крок, кінець) DO команда [параметри]***

Тут задана після ключового слова ***IN*** трійка ***(початок, крок, кінець)*** розкривається в послідовність чисел із заданими початком, кінцем і кроком збільшення. Так, набір ***(1,1,5)*** розкривається в ***(1 2 3 4 5)***, а набір ***(5, -1,1)*** замінюється на ***(5 4 3 2 1)***. Наприклад, в результаті виконання наступного командного файлу:

***@ECHO OFF***

***CLS***

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

змінна циклу ***%%f*** пробіжить значення від 1 до 5, і на екрані надрукуються п'ять чисел:

***1***

***2***

***3***

***4***

***5***

Числа, одержувані в результаті виконання циклу ***FOR /L***, можна використовувати в арифметичних обчисленнях. Розглянемо командний файл ***my.bat*** такого змісту:

***@ECHO OFF***

***CLS***

***FOR /L %%f IN (1,1,5) DO CALL :2 %%f***

***GOTO: EOF***

***:2***

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

***ECHO 10\*%1=%M%***

У третьому рядку в циклі відбувається виклик нового контексту файлу ***my.bat*** з поточним значенням змінної циклу ***%%f*** в якості параметра командного рядка, причому управління передається на мітку: 2. У шостому рядку змінна циклу множиться на десять, і результат записується в змінну M. Таким чином, в результаті виконання цього файлу виведеться наступна інформація:

***10\*1=10***

***10\*2=20***

***10\*3=30***

***10\*4=40***

***10\*5=50***

### Цикл FOR / F ... IN ... DO ...

Найпотужніші можливості (і одночасно самий заплутаний синтаксис) має команда: ***FOR*** з ключем ***/F***:

***FOR /F ["ключі"] %%змінна IN (набір)***

***DO команда [параметри]***

Тут параметр ***набір*** містить імена одного або декількох файлів, які по черзі відкриваються, читаються і обробляються. Обробка полягає в читанні файлу, розбитті його на окремі рядки тексту і виділення з кожного рядка заданого числа підрядків. Потім знайдений підрядок використовується в якості значення змінної при виконанні основного тіла циклу (заданої команди).

За замовчуванням ключ ***/F*** виділяє з кожного рядка файлу перше слово, очищене від оточуючих його прогалин. Порожні рядки в файлі пропускаються. Необов'язковий параметр ***"ключі"*** служить для перевизначення заданих за замовчуванням правил обробки рядків. Ключі є укладеним в лапки рядком, що містить наведені у табл. 3.2 ключові слова:

Табл. 3.2. Ключові слова для використання циклу ***FOR*** з ключем ***/F***

|  |  |
| --- | --- |
| **Ключ** | **Опис** |
| ***EOL=C*** | Визначення символу коментарів на початку рядка (допускається завдання тільки одного символу) |
| ***SKIP=N*** | Число пропускаються при обробці рядків на початку файлу |
| ***DELIMS=XXX*** | Визначення набору роздільників для заміни заданих за замовчуванням пробілу і знака табуляції |
| ***TOKENS=X,Y, M-N*** | Визначення номерів підрядків, що виділяються з кожного рядка файлу і переданих для виконання в тіло циклу |

При використанні ключа ***TOKENS=X,Y,M-N*** створюються додаткові змінні. Формат ***M-N*** представляє собою діапазон подстрок з номерами від ***M*** до ***N***. Якщо останній символ в рядку ***TOKENS*** є зірочкою, то створюється додаткова змінна, значенням якої буде весь текст, що залишився в рядку після обробки останнього підрядка.

Розберемо застосування цієї команди на прикладі пакетного файлу ***parser.bat***, який виробляє розбір файлу ***myfile.txt***:

***@ECHO OFF***

***IF NOT EXIST myfile.txt GOTO :NoFile***

***FOR /F "EOL=; TOKENS=2,3\* DELIMS =," %%i IN***

***(Myfile.txt) DO @ECHO %%i %%j %%k***

***GOTO :EOF***

***: NoFile***

***ECHO Не вдалося знайти файл myfile.txt!***

Тут у другому рядку проводиться перевірка наявності файлу ***myfile.txt***; в разі відсутності цього файлу виводиться попередження. Команда ***FOR*** в третьому рядку обробляє файл ***myfile.txt*** наступним чином:

Пропускаються всі рядки, які починаються з крапки з комою ***(EOL =;)***.

Другий і третій підрядки з кожного рядка передаються в тіло циклу, причому підрядки розділяються проміжками (за замовчуванням) і / або комами ***(DELIMS =,)***.

У тілі циклу змінна ***%%i*** використовується для другого підрядка, ***%%j*** - для третього, а ***%%k*** отримує всю решту підрядків після третього.

У нашому прикладі змінна ***%%i*** явно описана в інструкції ***FOR***, а змінні ***%%j*** і ***%%k*** описуються неявно за допомогою ключа ***TOKENS=***. Наприклад, якщо в файлі ***myfile.txt*** були записані такі три рядки:

***ААА ББББ ЧВВВ, ГГГГГ ДДДД***

***Еееее, жжжж зззз***

***; КККК ЛЛЛЛЛ МММММ***

то в результаті виконання пакетного файлу parser.bat на екран виведеться наступне:

***ББББ ЧВВВ ГГГГГ ДДДД***

***жжжж зззз***

Ключ ***TOKENS=*** дозволяє витягти з одного рядка файлу до 26 підрядків, тому заборонено використовувати імена змінних, починаються не з букв англійського алфавіту (***a-z***). Слід пам'ятати, що імена змінних ***FOR*** є глобальними, тому одночасно не може бути активно понад 26 змінних.

Команда ***FOR /F*** також дозволяє обробити окремий рядок. Для цього слід ввести потрібний рядок в лапках замість набору імен файлів в дужках. Рядок буде оброблений так, як ніби він взятий з файлу. Наприклад, файл такого змісту:

***@ECHO OFF***

***FOR /F "EOL=; TOKENS=2,3\* DELIMS=," %%i IN***

***("ААА ББББ ЧВВВ, ГГГГГ ДДДД") DO @ECHO %%i %%j %%k***

при своєму виконанні надрукує

***ББББ ЧВВВ ГГГГГ ДДДД***

Замість явного завдання рядка для розбору можна користуватися змінними середовища, наприклад:

***@ECHO OFF***

***SET M = ААА ББББ ЧВВВ, ГГГГГ ДДДД***

***FOR /F "EOL=; TOKENS=2,3\* DELIMS=,***

***"%% i IN ("%M%") DO @ECHO %%i %%j %%k***

Нарешті, команда ***FOR /F*** дозволяє обробити рядок виводу іншої команди. Для цього слід замість набору імен файлів в дужках ввести рядок виклику команди в апострофі (не в лапках!). Рядок передається для виконання інтерпретатора команд ***cmd.exe***, а виведення цієї команди записується в пам'ять і обробляється так, як ніби рядок виведення взятий з файлу. Наприклад, наступний командний файл:

***@ECHO OFF***

***CLS***

***ECHO Імена змінних середовища:***

***ECHO.***

***FOR /F "DELIMS =="%%i IN ('SET') DO ECHO %% i***

виведе перелік імен всіх змінних середовища, визначених в даний час в системі.

У циклі ***FOR*** допускається застосування тих самих синтаксичних конструкцій (операторів), що і для замінних параметрів (табл. 3.3).

Табл. 3.3. Синтаксичні конструкції, що можуть використані у циклі

|  |  |
| --- | --- |
| **Оператори** | **Опис** |
| ***%~Fi*** | Змінна ***%i*** розширюється до повного імені файлу |
| ***%~Di*** | Із змінної ***%i*** виділяється тільки ім'я диска |
| ***%~Pi*** | Із змінної ***%i*** виділяється тільки шлях до файлу |
| ***%~Ni*** | Із змінної ***%i*** виділяється тільки ім'я файлу |
| ***%~Xi*** | Із змінної ***%i*** виділяється розширення імені файлу |
| ***%~Si*** | Значення операторів N та X для змінної ***%i*** змінюється так, що вони працюють з коротким ім'ям файлу |

Якщо планується використовувати розширення підстановки значень в команді ***FOR***, то слід уважно підбирати імена змінних, щоб вони не перетиналися з позначеннями формату.

Наприклад, якщо ми знаходимося в каталозі ***C:​\Program Files\Far*** і запустимо командний файл такого змісту:

***@ECHO OFF***

***CLS***

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

то на екран виведуться повні імена всіх файлів  з розширенням txt:

***C:\Program Files\Far\Contacts.txt***

***C:\Program Files\Far\FarFAQ.txt***

***C:\Program Files\Far\Far\_Site.txt***

***C:\Program Files\Far\License.txt***

***C:\Program Files\Far\License.xUSSR.txt***

***C:\Program Files\Far\ReadMe.txt***

***C:\Program Files\Far\register.txt***

***C:\Program Files\Far\WhatsNew.txt***

# Лекція №4. «Введення до серверу сценаріїв WSH. Мови для програмування сценаріїв WSH. Мова JScript»

## Сервер сценаріїв WSH. Мови сценаріїв VBScript і JScript

У троьх попередніх лекціях було розглянуто мову командних файлів (мова командної оболонки), як в якості інструменту для автоматизації роботи підтримується у всіх версіях Windows. Однак за допомогою командного інтерпретатора ***cmd.exe*** важко написати будь-яку складну програму-сценарій (***script***): відсутня повноцінна інтерактивність, не можна безпосередньо працювати з робочим столом Windows і системним реєстром і т. д.

Для виправлення цієї ситуації компанією Microsoft був розроблений сервер сценаріїв ***Windows Script Host (WSH)***, за допомогою якого можна виконувати сценарії, написані, в принципі, будь-якою мовою (за умови, що для цього мови встановлений відповідний модуль (scripting engine), що підтримує технологію ActiveX Scripting). В якості стандартних мов підтримуються Visual Basic Script Edition (VBScript) і JScript.

Взагалі кажучи, принцип роботи сценаріїв, підтримуваних WSH, полягає у використанні об'єктів ActiveX, тому спочатку ми дуже коротко опишемо можливості самої технології ActiveX компанії Microsoft.

### Можливості технології ActiveX

Нагадаємо, що в Windows з самого початку для забезпечення обміну даними між додатками була розроблена технологія зв'язування та впровадження об'єктів (Object Linking and Embedding, OLE). Спочатку технологія OLE використовувалася для створення складених документів, а потім для вирішення більш загальної задачі - надання додатками один одному власних функцій (служб) і правильного використання цих функцій. Технологія, що дозволяє одному з додатків (клієнту автоматизації) викликати функції іншого додатка (сервера автоматизації) була названа OLE Automation. В основі OLE і OLE Automation лежить розроблена Microsoft базова "компонентна" технологія Component Object Model (COM). У загальних словах, компонентне програмне забезпечення - це спосіб розробки програм, при якому використовуються технології створення програмних модулів, подібні технологіям, застосовуваним для розробки апаратних засобів. Складні елементні схеми збираються з стандартизованих мікросхем, які мають чітко визначені документовані функції. Розробник може ефективно користуватися такими мікросхемами, не замислюючись про їх внутрішню структуру. У програмних компонентах, написаних на якій-небудь мові програмування, деталі реалізації використовуваних алгоритмів також приховані всередині компонента (об'єкта), а на поверхні знаходяться загальнодоступні інтерфейси, якими можуть користуватися і інші додатки, написані на тому ж або іншою мовою.

В даний час термін OLE використовується тільки з історичних причин. Замість нього Microsoft з 1996 року використовує новий термін - ActiveX, спочатку означав WWW (World Wide Web) компоненти (об'єкти), створені на базі технології COM.

Технологія ActiveX тривалий час була ключовою в продуктах Microsoft. Найбільш повне втілення вона знайшла в програмах Microsoft Office, Internet Explorer, Internet Information Service (IIS). У ці продукти для управління відповідними об'єктами автоматизації були вбудовані інтерпретатори спеціальних мов сценаріїв: VBScript (використовується в Microsoft Office, Internet Explorer, IIS) і JScript (використовується в Internet Explorer, IIS). Однак безпосередньо в операційній системі, поза цих продуктів, виконуватиме сценарії, написані на VBScript або JScript, було не можна.

Сервер сценаріїв WSH є потужним інструментом, що надає єдиний інтерфейс (об'єктну модель) для спеціалізованих мов (VBScript, JScript, PerlScript, REXX, TCL, Python і т. П.), Які, в свою чергу, дозволяють використовувати будь-які зовнішні об'єкти ActiveX. За допомогою WSH сценарії можуть бути виконані безпосередньо в операційній системі Windows, без вбудовування в HTML-сторінки.

### Призначення і основні властивості WSH

WSH ставить мінімальні вимоги до обсягу оперативної пам'яті, і є дуже зручним інструментом для автоматизації повсякденних завдань користувачів і адміністраторів операційної системи Windows. Використовуючи сценарії WSH, можна безпосередньо працювати з файлової системою комп'ютера, а також керувати роботою інших програм (серверів автоматизації). При цьому можливості сценаріїв обмежені тільки засобами, які надають доступні сервери автоматизації.

Перерахуємо лише найбільш очевидні завдання, для автоматизації яких прекрасно підходять сценарії WSH:

* Організація резервного копіювання на мережевий сервер файлів з локальної машини, які відбираються за будь-яким критерієм.
* Швидка зміна конфігурації робочого столу Windows в залежності від завдань, що виконуються.
* Автоматичний запуск програм Microsoft Office, створення там складних складових документів, роздруківка цих документів і закриття додатків.
* Управління роботою додатків, які не є серверами автоматизації, за допомогою посилки в ці додатки натискань клавіш.
* Підключення і відключення мережевих ресурсів (дисків і принтерів).
* Створення складних сценаріїв реєстрації для користувачів.
* Виконання завдань адміністрування локальної мережі (наприклад, додавання або видалення користувачів).

### Створення і запуск найпростіших сценаріїв WSH

Найпростіший WSH-сценарій, написаний на мові ***JScript*** або ***VBScript*** - це звичайний текстовий файл з розширенням ***js*** або vbs відповідно, створити його можна в будь-якому текстовому редакторі, здатному зберігати документи в форматі "Тільки текст".

Розмір сценарію може змінюватися від однієї до тисяч рядків, граничний розмір обмежується лише максимальним розміром файлу у відповідній файлової системи.

Як перший приклад створимо JScript-сценарій, що виводить на екран діалогове вікно з написом "Привіт!". Для цього достатньо c допомогою, наприклад, стандартного Блокнота Windows (notepad.exe) створити файл ***First.js***, що містить всього один рядок:

***WScript.Echo("Привіт!");***

Той же самий сценарій на мові ***VBScript***, природно, відрізняється синтаксисом і виглядає наступним чином:

***WScript.Echo "Привіт!"***

Незважаючи на те, що для роботи цих двох сценаріїв достатньо всього одного рядка, бажано відразу привчити себе до додавання в початок файлу інформації про що знаходиться в ньому сценарії: ім'я файлу, який використовується мова, короткий опис виконуваних дій. Мовою ***JScript*** така інформація, оформлена у вигляді коментарів, може виглядати наступним чином:

***/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/***

***/ \* Ім'я: First.js \* /***

***/ \* Мова: JScript \* /***

***/ \* Опис: Виведення на екран вітання \* /***

***/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/***

Мовою ***VBScript*** те ж саме виглядає наступним чином:

***'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\****

***'Ім'я: First.vbs***

***'Мова: VBScript***

***'Опис: Виведення на екран вітання***

***'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\****

Для запуску сценаріїв WSH існує кілька способів.

Можна виконати сценарій з командного рядка за допомогою консольної версії WSH cscript.exe. Наприклад, щоб запустити сценарій, записаний у файлі ***C:\Script\First.js***, потрібно завантажити командне вікно і виконати в ньому команду

***cscript C:\Script\First.js***

В результаті виконання цього сценарію в командне вікно виведеться рядок ***"Привіт!"***

Виконання сценарію контролюється за допомогою параметрів командного рядка для ***cscript.exe*** (табл. 4.1), які вмикають або вимикають різні опції WSH (всі ці параметри починаються з символів ***//***).

Табл. 4.1. Параметри командного рядка ***cscript.exe***

|  |  |
| --- | --- |
| **Параметр** | **Опис** |
| ***//I*** | Вимикає пакетний режим (за замовчуванням). При цьому на екран виводитимуться всі повідомлення про помилки в сценарії |
| ***//B*** | Включає пакетний режим. При цьому повідомлення про помилки в сценарії на екран не будуть виводитись |
| ***//Т:nn*** | Задає тайм-аут в секундах, тобто сценарій буде виконуватись ***nn*** секунд, після чого процес перерветься. За замовчуванням час виконання не обмежений |
| ***//logo*** | Виводить (за замовчуванням) перед виконанням сценарію інформацію про версію та розробника WSH |
| ***//nologo*** | Пригнічує висновок інформації про версію та розробника WSH |
| ***//H:CScipt*** або ***//H:WScript*** | Робить ***cscript.exe*** або wscript.exe додатком для запуску сценаріїв за замовчуванням. Якщо ці характеристики не вказані, то за замовчуванням мається на увазі ***wscript.exe*** |
| ***//S*** | Зберігає установки командного рядка для поточного користувача |
| ***//?*** | Виводить вбудовану підказку для параметрів командного рядка |
| ***//E:engine*** | Виконує сценарій за допомогою модуля, заданого параметром ***engine*** |
| ***//D*** | Вмикає налагоджувач |
| ***//X*** | Виконує програму у налагоджувачі |
| ***//Job:<JobID>*** | Запускає завдання з індексом ***JoblD*** з многозадачного WS-файлу |

Наприклад, команда

***cscript //В my.js /а /b***

запустить сценарій ***my.js*** в пакетному режимі, при цьому ***/а*** і ***/b*** будуть являтись параметрами цього сценарію, а ***//B*** - параметром додатку ***cscript.exe***.

Сценарій можна виконати з командного рядка за допомогою (віконної) графічної версії WSH ***wscript.exe***. Для нашого прикладу в цьому випадку потрібно виконати команду

***wscript C:\Script\First.js***

Тоді в результаті виконання сценарію на екрані з'явиться потрібне нам діалогове вікно.

Таким чином, при запуску сценарію в консольному режимі, висновок текстової інформації відбувається в стандартний вихідний потік (на екран), при запуску в графічному режимі - в діалогове вікно.

Для запуску сценарію за допомогою пункту ***Виконати (Run)*** меню ***Пуск (Start)***, достатньо написати повне ім'я цього сценарію в поле ***Відкрити (Open)*** і натиснути кнопку ***OK***. У цьому випадку за замовчуванням сценарій буде виконано за допомогою ***wscript.exe***, тобто виведення інформації буде вестися в графічне діалогове вікно.

Найпростішим є запуск сценарію в вікнах Провідника Windows або на робочому столі - досить просто виконати подвійне клацання мишею на імені файлу зі сценарієм або на його значку (аналогічно будь-якого іншого виконуваного файлу). При цьому, як і в разі запуску за допомогою меню ***Пуск (Start)***, сценарій за замовчуванням виконується за допомогою ***wscript.exe***.

**Мови VBScript і JScript для сценаріїв WSH**

Як вже зазначалося вище, в принципі сценарії ***WSH*** можна писати на будь-якій мові, що підтримує технологію ***ActiveX Scripting***. Однак для всіх таких мов, крім ***VBScript*** і ***JScript***, в системі доведеться додатково встановлювати ті чи інші модулі (бібліотеки) підтримки. Тому найчастіше при роботі з ***WSH*** користуються саме мовами ***VBScript*** або ***JScript*** - сценарії на цих мовах гарантовано працюють на будь-якій машині з операційною системою Windows.

За синтаксису і стилю програмування ці дві мови сильно відрізняються один від одної. Мова ***JScript*** - це розроблена Microsoft інтерпретовона об'єктно-орієнтована мова сценаріїв, який спочатку призначалась для створення динамічних HTML-сторінок. Відзначимо, що ***JScript*** не є урізаною версією будь-якої іншої мови програмування, хоча по синтаксису вона схожа на мови Java і C. Мова ***VBScript*** (Visual Basic Script Edition) - це полегшена версія мови Microsoft Visual Basic, тому для тих, хто програмував на Visual Basic або VBA (Visual Basic for Application) мова ***VBScript*** виявиться дуже знайомою.

Яку саме з мов використовувати - в більшій мірі справа смаку і звички. Якщо ви писали на C або Java, то, швидше за все, зупинитесь на ***JScript***; якщо ви програмуєте на Visual Basic або VBA (Visual Basic for Application), то дуже знайомим вам виявиться мова ***VBScript***.

Відзначимо також, що при виборі мови для написання сценарію, що звертається до зовнішніх об'єктів (а це робиться практично в кожному сценарії Windows), на користь ***VBScript*** можна навести кілька додаткових аргументів.

По-перше, ***VBScript*** дозволяє безпосередньо в циклі ***For Each ... Next*** перебирати елементи колекцій (такі колекції часто є властивостями зовнішніх об'єктів), а в ***JScript*** для цього доводиться використовувати допоміжний об'єкт ***Enumerator*** і цикл ***for*** з умовою завершення і оператором ітерації. Розглянемо наступний приклад. Припустимо, що змінна ***Folder*** є об'єктом, відповідним кореневого каталогу диска ***C:***, тобто ***Folder*** являє собою колекцію, яка містить об'єкти-файли, що знаходяться в кореневому каталозі. Тоді перебір всіх цих файлів в ***VBScript*** організовується таким чином:

***'Створюємо колекцію Files всіх файлів в кореневому каталозі диска C:***

***Set Files = Folder.Files***

***'Перебираємо всі елементи колекції Files***

***For Each File In Files***

***'Виділяємо ім'я файлу для поточного елемента File колекції***

***s = s & File.Name & vbCrLf***

***Next***

Аналогічний код на JScript виглядає так:

***// Створюємо колекцію файлів***

***Files = new Enumerator(Folder.Files);***

***// Цикл по всіх файлів***

***for (; !Files.atEnd(); Files.moveNext())***

***// Додаємо рядок з ім'ям файлу***

***s + = Files.item().Name+"\n";***

***// Виводимо отримані рядки на екран***

По-друге, ***VBScript*** дозволяє безпосередньо викликати методи об'єктів ***WMI***, в той час як в ***JScript*** ці методи доводиться викликати за допомогою спеціального об'єкта ***SWbemObject***, а також об'єктів ***InParam*** (входять параметри для виклику методу) і ***OutParam*** (параметри, що формуються після виконання методу).

Нарешті, переважна більшість прикладів сценаріїв, які можна знайти в документації Microsoft або мережі Інтернет, написані на ***VBScript***, їх можна використовувати в своїх розробках без змін, не витрачаючи час на переклад коду на іншу мову.

Детальний розгляд синтаксису мов ***VBScript*** і ***JScript*** виходить за рамки даного курсу; нижче в прикладах сценаріїв ми будемо використовувати ***JScript***.

***Основні відомості про мову Jscript***

Мова **JScript** - це розроблена Microsoft об'єктно-орієнтована мова сценаріїв, що інтерпретується, яка спочатку призначалась для створення динамічних HTML-сторінок. Відзначимо, що **JScript** не є урізаною версією будь-якої іншого мови програмування, хоча по синтаксису вона схожа на мови **Java** і **С**. Далі ми коротко розглянемо ті можливості і властивості JScript, які можуть знадобитися при складанні сценаріїв, що виконуються за допомогою WSH.

У сценаріях **JScript**, як і в будь-якій іншій мові програмування, можна використовувати змінні, звертаючись до них по імені. При цьому змінні можуть бути як глобальними (доступними з будь-якого місця сценарію), так і локальними (область дії обмежується функцією, в якій вони визначені).

Хорошим тоном вважається попереднє оголошення використовуваних змінних за допомогою ключового слова ***var***, хоча це є обов'язковою умовою тільки для локальних змінних, визначених у функціях. Приклад оголошення змінної має вигляд:

***var MyVariable;***

При оголошенні тип змінної явно не вказується (як це робиться, наприклад, в мовах **С** або **Pascal**). Певний тип змінної присвоюється тільки тоді, коли в неї записується якесь значення. Мова **JScript** є регістру залежною, тобто імена ***MyVariable*** і ***myvariabie*** представляють різні змінні. Крім цього, при виборі імен змінних слід дотримуватися наступних правил:

* ім'я змінної повинно починатися з літери або з символів "**\_**", "***$***" і може складатися тільки з букв, цифр, а також символів "***\_***", "***$***";
* ім'я змінної не повинно збігатися з зарезервованими ключовими словами мови **JScript**.

У **JScript** підтримуються шість типів даних, головними з яких є числа, рядки, об'єкти і логічні дані. Два останні типи - це ***null*** (порожній тип) і ***undefined*** (невизначений тип).

Функції в **JScript**, як і в інших алгоритмічних мовах, дозволяють об'єднати кілька операцій під одним ім'ям. У випадку необхідності функція може бути викликана з будь-якого місця сценарію. У сценаріях **JScript** підтримуються два види функцій: вбудовані функції і функції користувача, які ми пишемо самі.

Функції, визначені користувачем, можуть перебувати в будь-якому місці сценарію і мати довільне число параметрів (аргументів). Загальний вигляд визначення функції має вигляд:

***function Ім’я\_функції([параметр1] [, параметр2] [..., параметри] ) {***

***…***

***тіло функції***

***…***

***[return значення;]***

***}***

Ключове слово ***return*** дозволяє функції повернути значення будь-якого допустимого типу. Наприклад, наведена нижче функція ***MyFunction*** повертає ***true***, якщо обидва її аргументи менше 10:

***function MyFunction(x,у) {***

***if ((х<10) && (у<10))***

***return true***

***else***

***return false;***

***}***

***Вбудовані об’єкти (класи)***

Як уже було зазначено вище, **JScript** є об'єктно-орієнтованою мовою, тому робота з рядками, датою і часом, а також такими структурами, як масиви і колекції, здійснюється за допомогою відповідних вбудованих об'єктів. Крім цього, математічні обчислення виконуються також за допомогою спеціального вбудованого об'єкта. У табл. 4.2 описані деякі об'єкти, які можуть бути корисними при створенні сценаріїв за допомогою WSH.

Табл. 4.2. Деякі вбудовані об’єкти **Jscript**

|  |  |
| --- | --- |
| **Об'єкт** | **Опис** |
| ***Array*** | Створення і робота з масивами даних довільного типу |
| ***Date*** | Робота з даними, що містять дату або час |
| ***Enumerator*** | Робота з колекціями даних довільного типу |
| ***Math*** | Виконання математичних обчислень |
| ***String*** | Робота з текстовими рядками |

Для того щоб в сценарії використовувати вбудований об'єкт, необхідно створити змінну, за допомогою якої можна буде отримати доступ до властивостей і методів цього об'єкта. Для створення більшості змінних такого виду застосовується оператор new і спеціальна функція - конструктор потрібного об'єкту. Назва конструктора завжди збігається з назвою відповідного вбудованого об'єкта. Наведемо приклад створення об'єктів ***Date*** і ***Array***:

***var d;***

***d = new Date();***

***var a;***

***a = new Array(10);***

Відзначимо, що об'єкти ***String*** можна створювати, просто записуючи в лапках значення рядка:

***var s;***

***s = "Привіт!";***

**Об’єкт Array**

Новий об'єкт вбудованого класу ***Array*** можна створити за допомогою оператора ***new*** наступними способами:

* ***new Array()*** - створення масиву нульової довжини;
* ***new Array(N)*** - створення масиву довжини ***N***;
* ***new Array(a0, a1, ..., aN)*** - створення масиву довжини ***N+1*** з елементами ***а0, ai, ..., aN***.

Наприклад:

***var Al, A2, A3;***

***Al = new Array();***

***A2= new Array(3);***

***A3 = new Array(0,"Рядок",2.5};***

Нумерація елементів в масивах завжди починається з нуля. Після того як масив створений і проініціалізований, звертатися до його елементів можна за допомогою звичайного оператора індексації ***[]***, наприклад:

***А3[1] = А3[0] + A3[2];***

Довжину масиву, тобто кількість елементів, які містяться у ньому, можна дізнатися за допомогою властивості ***length*** об'єкта ***Array***. Для того щоб динамічно змінити довжину масиву (зменшити або збільшити), достатньо просто записати відповідне значення у властивість ***length***:

***var A,***

***А = new Array(1,2,3,4,5); //Довжина масиву А дорівнює 5***

***A.length = 3; //Тепер довжина масиву А дорівнює 3***

**Об’єкт Date**

Для створення нового об'єкта вбудованого класу ***Date*** використовується один з трьох конструкторів.

Конструктор першого виду дозволяє створити об'єкт, в якому зберігається інформація про поточну дату і час:

***var d;***

***d = new Date();***

Тут час задається за Гринвічем, з використанням часу Universal Cooidmated Time (UCT)

Конструктор другого виду має єдиний параметр:

***var d;***

***d = new Date(nMilliseconds);***

Параметр ***nMilliseconds*** задає дату в мілісекундах, рахуючи від 1 січня 1970 року

Конструктор третього виду призначений для роздільного завдання компонентов дати і має такий вигляд

***var d;***

***d = New Date (year, month, date [, hours [, mm [, sec [, ms]]]]);***

**Об’єкт Enumerator**

За допомогою об'єкта ***Enumerator*** можна отримати доступ до будь-якого елементу колекції (в **VBScript** для цього служить цикл ***For .. Each***). Колекцією у мові **JScript** називається множина елементів, яка відрізняється від масиву тим, що до елементів колекції можна отримати прямий доступ за допомогою індексів - можна тільки переміщати вказівник поточного елемента на самий перший або наступний щодо поточного елемент.

Для створення нового об'єкта вбудованого класу ***Enumerator*** використовується конструктор наступного виду:

***var e;***

***е = new Enumerator(collection);***

Тут параметр ***collection*** вказує на колекцію, для доступу до елементів якої і створюється об'єкт класу ***Enumerator***. Самі колекції зазвичай є властивостями інших об'єктів. Методи об'єкта ***Enumerator*** представлені в табл. 4.3 (властивостей у цього об'єкєкта немає).

Табл. 4.3. Методи об’єкта ***Enumerator***

|  |  |
| --- | --- |
| **Метод** | **Опис** |
| ***atEnd()*** | Повертає ***true***, якщо покажчик поточного елемента знаходиться на елементі, наступного за останнім екземпляром колекції, або колекція порожня, або поточний елемент не визначений. В іншому випадку повертає ***false*** |
| ***item()*** | Повертає значення поточного елемента колекції. Якщо коллекція порожня або поточний елемент не визначений, повертається невизначене значення ***undefined*** |
| ***moveFirst()*** | Переміщує вказівник на перший елемент колекції. Якщо в колекції немає елементів, поточний елемент приймає невизначене значення ***undefined*** |
| ***moveNext()*** | Переміщує вказівник на наступний елемент колекції. Якщо перед застосуванням цього методу вказівник знаходився на останньому елементі колекції, або колекція порожня, поточний елемент приймає невизначене значення ***undefined*** |

***var fso, s, n, e, x;***

//Створення об’єкта FileSystemObject

***fso = WScript.CreateObject("Scripting.FileSystemObject");***

***//Створення об’єкта Enumerator для доступу до колекції fso.Drives***

***е = new Enumerator(fso.Drives);***

***s = "";***

***// Цикл для перегляду всіх елементів колекції***

***for (; !e.atEnd(); e.moveNext()) {***

***//Витягнення елементів колекції***

***х = е.item() ;***

***s = s + x.DriveLetter;***

***s += " - ";***

***if (x.DriveType == 3)***

***n = x.ShareName;***

***else***

***if (x.IsReady)***

***n = x.VolumeName;***

***else***

***n = "[Пристрій не готовий]";***

***s += n + "\n";***

***}***

Тут за допомогою методу ***CreateObject*** об'єкта ***WScript*** (цей об'єкт буде описаний нижче) створюється екземпляр об'єкта ***FileSystemObject*** (змінна ***fso***), за допомогою якого можна отримати доступ до файлової системи комп'ютера (даний об’єкт буде описаний у наступних лекціях). Властивість ***Drives*** цього об'єкта є колекцією, яка містить відомості про всі доступні пристрої (Диски).

У циклі ми формуємо рядок ***s***, в якій будуть записані букви всіх дисків комп'ютера і імена цих дисків (в разі локального диска - це мітка диска, а в разі мережевого - ім'я диска в UNC-форматі).

## Власна об'єктна модель WSH

Перейдемо до опису власної об'єктної моделі ***Windows Script Host***. За допомогою внутрішніх об'єктів ***WSH*** з сценаріїв можна виконувати такі основні завдання:

* виводити інформацію в стандартний вихідний потік (на екран) або в діалогове вікно Windows;
* читати дані з стандартного вхідного потоку (тобто вводити дані з клавіатури) або використовувати інформацію, виведену іншою командою;
* використовувати властивості і методи зовнішніх об'єктів, а також обробляти події, які генеруються цими об'єктами;
* запускати нові незалежні процеси або активізувати вже наявні;
* запускати дочірні процеси з можливістю контролю їх стану і доступу до їх стандартним вхідним і вихідним потокам;
* працювати з локальною мережею: визначати ім'я зареєстрованого користувача, підключати мережеві диски та принтери;
* переглядати і змінювати змінні середовища;
* отримувати доступ до спеціальних папках Windows;
* створювати ярлики Windows;
* працювати з системним реєстром.

В WSH входять перераховані нижче об'єкти:

* ***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 дані з клавіатури і виводити інформацію на екран або у вікно Windows.
* ***WshArguments***. Забезпечує доступ до всіх параметрів командного рядка запущеного сценарію або ярлика Windows.
* ***WshNamed***. Забезпечує доступ до іменних параметрів командного рядка запущеного сценарію.
* ***WshUnnamed***. Забезпечує доступ до безіменних параметрів командного рядка запущеного сценарію.
* ***WshShell***. Дозволяє запускати незалежні процеси, створювати ярлики, працювати зі змінними середовища, системним реєстром і спеціальними папками Windows.
* ***WshSpecialFolders***. Забезпечує доступ до спеціальних папок Windows.
* ***WshShortcu***t. Дозволяє працювати з ярликами Windows.
* ***WshUrlShortcut***. Призначений для роботи з ярликами мережевих ресурсів.
* ***WshEnvironment***. Призначений для перегляду, зміни і видалення змінних середовища.
* ***WshNetwork***. Використовується при роботі з локальною мережею: містить мережеву інформацію для локального комп'ютера, дозволяє підключати мережеві диски та принтери.
* ***WshScriptExec***. Дозволяє запускати консольні додатки в якості дочірніх процесів, забезпечує контроль стану цих додатків і доступ до їх стандартних вхідних і вихідних потоків.
* ***WshController***. Дозволяє запускати сценарії на віддалених машинах.
* ***WshRemote***. Дозволяє управляти сценарієм, запущеним на віддаленій машині.
* ***WshRemoteError***. Використовується для отримання інформації про помилку, що виникла в результаті виконання сценарію, запущеного на віддаленій машині.

Крім цього, є об'єкт ***FileSystemObject***, що забезпечує доступ до файлової системи комп'ютера (цей об'єкт буде детально розглянуто в наступній лекції).

### Об'єкт WScript

Властивості об'єкта ***WScript*** дозволяють отримати повний шлях до що використовується сервера сценаріїв (***wscript.exe*** або ***cscript.exe***), параметри командного рядка, з якими запущений сценарій, режим його роботи (інтерактивний або пакетний). Крім цього, за допомогою властивостей об'єкта ***WScript*** можна виводити інформацію в стандартний вихідний потік і читати дані з стандартного вхідного потоку. Також ***WScript*** надає методи для роботи всередині сценарію з об'єктами автоматизації і виведення інформації на екран (в текстовому режимі) або в вікно Windows.

Відзначимо, що в сценарії ***WSH*** об'єкт ***WScript*** можна використовувати відразу, без будь-якого попереднього опису або створення, так як його екземпляр створюється сервером сценаріїв автоматично. Для використання ж усіх інших об'єктів потрібно використовувати або метод ***CreateObject***, або певну властивість іншого об'єкта.

Властивості об'єкта WScript представлені у табл. 4.4.

Табл. 4.4. Властивості об’єкта ***WScript***

|  |  |
| --- | --- |
| **Властивість** | **Опис** |
| ***Arguments*** | Містить вказівник на колекцію ***WshArguments***, що містить параметри командного рядка для виконуваного сценарію |
| ***FullName*** | Містить повний шлях до виконуваного файлу сервера сценаріїв (в Windows зазвичай це ***C:\WINDOWS\SYSTEM32\CSCRIPT.EXE*** або ***C:\WINDOWS\SYSTEM32\WSCRIPT.EXE***) |
| ***Name*** | Містить назву об'єкта ***WScript*** (Windows Script Host) |
| ***Path*** | Містить шлях до каталогу, в якому знаходиться cscript.exe або ***wscript.exe*** (в Windows зазвичай це ***C:\WINDOWS\SYSTEM32***) |
| ***ScriptFullName*** | Містить повний шлях до запущеного сценарію |
| ***ScriptName*** | Містить ім'я запущеного сценарію |
| ***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*** | Містить версію WSH |

Опишемо більш детально деякі властивості об'єкта ***WScript***.

### Властивість Arguments

У наступному прикладі на екран виводяться всі параметри командного рядка, з якими був запущений сценарій.

***var e, s, objArgs, Arg***

***objArgs = WScript.Arguments; //Створюємо об'єкт WshArguments***

***e = new Enumerator(objArgs);***

***s=””;***

***for (var i=0; i<objArgs.length; i++) {***

***s+=objArgs(i)+“ “; //Формуємо рядок зі значеннями аргументів***

***}***

***WScript.Echo(s); //Виводимо сформований рядок***

### Властивості StdErr, StdIn, StdOut

Доступ до стандартних вхідних і вихідних потоків за допомогою властивостей ***StdIn***, ***StdOut*** і ***StdErr*** можна отримати тільки в тому випадку, якщо сценарій запускався в консольному режимі за допомогою ***cscript.exe***. Якщо сценарій був запущений за допомогою ***wscript.exe***, то при спробі звернутися до цих властивостей виникне помилка ***"Invalid Handle"***.

Працювати з потоками ***StdOut*** і ***StdErr*** можна за допомогою методів ***Write***, ***WriteLine***, ***WriteBlankLines***, а з потоком ***StdIn*** - за допомогою методів ***Read***, ***ReadLine***, ***ReadAll***, ***Skip***, ***SkipLine***. Ці методи коротко описані у табл. 4.5.

Табл. 4.5. Методи для роботи зі стандартними потоками

|  |  |
| --- | --- |
| **Метод** | **Опис** |
| ***Read(n)*** | Зчитує з потоку ***StdIn*** заданий параметром ***n*** число символів і повертає отриманий рядок |
| ***ReadAll()*** | Читає символи з потоку ***StdIn*** до тих пір, поки не зустрінеться символ кінця файлу ASCII 26 (***<Ctrl>+<Z>***), і повертає отриманий рядок |
| ***ReadLine()*** | Повертає рядок, зчитану з потоку ***StdIn*** |
| ***Skip(n)*** | Пропускає при читанні з потоку ***StdIn*** заданий параметром n число символів |
| ***SkipLine()*** | Пропускає цілий рядок при читанні з потоку ***StdIn*** |
| ***Write(string)*** | Записує в потік ***StdOut*** або ***StdErr*** рядок ***string*** (без символу кінця рядка) |
| ***WriteBlankLines(n)*** | Записує в потік ***StdOut*** або ***StdErr*** заданий параметром n число порожніх рядків |
| ***WriteLine(string)*** | Записує в потік ***StdOut*** або ***StdErr*** рядок string (разом з символом кінця рядка) |

Нагадаємо, що операційна система Windows підтримує механізм конвеєризації (символ "***|***" в командному рядку). Цей механізм робить можливим передачу даних від однієї програми до іншої. Таким чином, використовуючи стандартні вхідні і вихідні потоки, можна зі сценарію обробляти рядки виведення іншої програми або перенаправляти виведені сценарієм дані на вхід програм-фільтрів (***FIND*** або ***SORT***). Наприклад, наступна команда буде сортувати рядки виведення сценарію ***example.js*** і виводити їх в файл ***sort.txt***:

***cscript //Nologo example.js | sort > sort.txt***

Опція ***//Nologo*** тут потрібна для того, щоб в файл sort.txt не потрапляли рядки з інформацією про розробника та номер версії WSH.

Крім цього, за допомогою методів, які працюють з вхідним потоком ***StdIn***, можна організовувати діалог з користувачем, тобто створювати інтерактивні сценарії. Відповідний приклад приведений далі.

***//Виводимо рядок на екран***

***WScript.StdOut.Write("Введіть рядок:");***

***//Прочитуємо рядок***

***s = WScript.StdIn.ReadLine();***

***//Виводимо рядок на екран***

***WScript.StdOut.WriteLine("Ви ввели число" + s);***

### Методи об'єкта WScript

Об'єкт ***WScript*** має кілька методів, які описані в табл. 4.6.

Табл. 4.6. Методи об’єкта ***WScript***

|  |  |
| --- | --- |
| **Метод** | **Опис** |
| ***CreateObject(strProgID [,strPrefix])*** | Створює об'єкт, заданий параметр ***strProgID*** |
| ***ConnectObject(strObject, strPrefix)*** | Встановлює з'єднання з об'єктом ***strObject***, що дозволяє писати функції-обробники його подій (імена цих функцій повинні починатися з префікса ***strPrefix***) |
| ***DisconnectObject(об'єкт)*** | Від'єднує об'єкт ***OBJ***, зв'язок з якими був попередньо встановлений в сценарії |
| ***Echo([Arg1] [, Arg2] [...])*** | Виводить текстову інформацію на консоль або в діалогове вікно |
| ***GetObject(strPathname [, strProgID], [strPrefix])*** | Активізує об'єкт автоматизації, що визначається заданим файлом (параметр ***strPathName***) або об'єкт, заданий параметром ***strProgID*** |
| ***Exit([intErrorCode])*** | Перериває виконання сценарію з заданим параметром ***intErrorCode*** кодом виходу. Якщо параметр не заданий ***intErrorCode***, то об'єкт ***WScript*** встановить код виходу рівний нулю |
| ***Sleep(IntTime)*** | Припиняє виконання сценарію (переводить його в неактивному стані) на заданому параметрі ***IntTime*** число мілісекунд |

Наведемо додаткові пояснення і приклади використання для методів, наведених в таблиці.

**Метод CreateObject**

Строковий параметр ***strProgID***, що вказується в методі ***CreateObject*** називається програмний ідентифікатор об'єкт (***Programmic ProgID*** ідентифікатора).

Якщо зазначений необов'язковий параметр ***strPrefix***, то після створення об'єкта в сценарії можна обробляти події, що виникають в цьому об'єкті (природно, якщо об'єкт надає інтерфейси для зв'язку з цими подіями). Коли об'єкт повідомляє про виникнення певної події, сервер сценарії викликає функцію, ім'я якої складається з префікса ***strPrefix*** і ім'я цієї події. Наприклад, якщо в якості ***strPrefix*** зазначено "***MYOBJ\_***" а об'єкт повідомляє про виникнення події "***OnBegin***," то буде запущена функція "***MYOBJ\_OnBegin***" яка повинна бути описана в сценарії.

У наступному прикладі метод ***CreateObject*** використовується для створення об'єкта ***WshNetwork***:

***var WshNetwork = WScript.CreateObject("WScript.Network");***

Відзначимо, що об'єкти автоматизації із сценаріїв можна створювати і без допомоги ***WSH***. У ***JScript*** для цього використовується об'єкт ***ActiveXObject***, наприклад:

***var WshNetwork = new ActiveXObject("WScript.Network");***

У ***VBScript*** для створення об'єктів може використовуватися спеціальна функція CreateObject, наприклад:

***Set WshNetwork = CreateObject("WScript.Network");***

Однак організувати в сценарії обробку подій створюваного об'єкта можна тільки при використанні методу ***WScript.CreateObject***.

**Метод ConnectObject**

Об'єкт, з'єднання з яким здійснюється за допомогою методу ***Connect*** об'єкта, повинен надавати інтерфейс до своїх подій.

У наступному прикладі в змінної ***MyObject*** створюється абстрактний об'єкт "***SomeObject***" потім зі сценарію викликається метод ***SomeMetod*** цього об'єкта. Після цього встановлюється зв'язок зі змінною ***MyObject*** і задається префікс "***MyEvent***" для процедур обробки події цього об'єкта. Якщо в об'єкті виникне подія з ім'ям "подія", то буде викликана функція ***MyEvent\_Event***. Метод ***DisconnectObject*** об'єкта ***WScript*** виробляє від'єднання об'єкта ***MyObject***.

***var MyObject = WScript.CreateObject("Деякий об'єкт");***

***MyObject.someMethod();***

***WScript.ConnectObject(MyObject, "MyEvent");***

***FunctionfMyEvent\_Event(strName) {***

***WScript.Echo(strName);***

***}***

***WScript.DisconnectObject(MyObject);***

**Метод Echo**

Параметри ***arg1, arg2, ...*** методу ***Echo*** задають аргументи для виведення. Якщо сценарій був запущений за допомогою ***wscript.exe***, то метод ***Echo*** направляє виведення в діалоговому вікні, якщо ж для виконання сценарію застосовується ***cscript.exe***, то виведення буде направлено на екран (консоль). Кожен з аргументів при виведенні буде розділений проміжком. У разі використання ***cscript.exe*** виведення всіх аргументів буде завершено символом нового рядка. Якщо в методі ***Echo*** не заданий жоден аргумент, то буде надрукований порожній рядок.

Наприклад, після виконання сценарію ***EchoExample.js*** за допомогою ***cscript.exe*** на екрані буде виведена порожній рядок, три числа і рядок тексту.

***WScript.Echo(); //Виводимо порожній рядок***

***WScript.Echo(1,2,3); //Виводимо числа***

***WScript.Echo("Привіт!"); //Виводимо рядок***

**Метод Sleep**

У наступному прикладі сценарій перекладається в неактивний стан на 5 секунд:

***WScript.Echo("Сценарій запущений, відпочиває ...");***

***WScript.Sleep(5000);***

***WScript.Echo("Виконання сценарію завершено");***

Метод Sleep необхідно застосовувати при асинхронної роботі сценарію і будь-якої іншої задачі, наприклад, при імітації натискань клавіш в активному вікні за допомогою методу ***WshShell.SendKeys***.

## Лекція №5. «Програмування сценаріїв WSH для роботи з файловою системою Windows»

## Сценарії WSH для доступу до файлової системи. Об'єктна модель FileSystemObject

Сценарії WSH дозволяють отримати повний доступ до файлової системи комп'ютера, на відміну від JScript- або VBScript-сценаріїв, впроваджених в HTML-сторінки, де в залежності від рівня безпеки, який встановлюється в настройках браузера, ті чи інші операції можуть бути заборонені.

### Об'єкти для основних операцій з файлової системою

Для роботи з файловою системою з сценаріїв WSH призначені 8 об'єктів, головним з яких є ***FileSystemObject***. За допомогою методів об'єкта ***FileSystemObject*** можна виконувати такі основні дії:

* копіювати або переміщати файли і каталоги;
* видаляти файли і каталоги;
* створювати каталоги;
* створювати або відкривати текстові файли;
* створювати об'єкти ***Drive***, ***Folder*** і ***File*** для доступу до конкретного диску, каталогу або файлу відповідно.

За допомогою властивостей об'єктів ***Drive***, ***Folder*** і ***File*** можна отримати детальну інформацію про ті елементи файлової системи, з якими вони асоційовані. Об'єкти ***Folder*** і ***File*** також надають методи для маніпулювання файлами і каталогами (створення, видалення, копіювання, переміщення); ці методи в основному копіюють відповідні методи об'єкта ***FileSystemObject***.

Крім цього є три об'єкти-колекції: ***Drives***, ***Folders*** і ***Files***. Колекція ***Drives*** містить об'єкти ***Drive*** для всіх наявних в системі дисків, ***Folders*** - об'єкти ***Folder*** для всіх підкаталогів заданого каталогу, ***Files*** - об'єкти ***File*** для всіх файлів, що знаходяться всередині певного каталогу.

Нарешті, з сценарію можна читати інформацію з текстових файлів і записувати в них дані. Методи для цього надає об'єкт ***TextStream***.

У табл. 5.1 коротко описано, які саме об'єкти, властивості і методи можуть знадобитися для виконання найбільш часто використовуваних файлових операцій.

Табл. 5.1. Виконання основних файлових сценаріїв

|  |  |
| --- | --- |
| **Операція** | **Використовувані об'єкти, властивості і методи** |
| Отримання відомостей про певний диск (тип файлової системи, мітка тому, загальний обсяг і кількість вільного місця і т.д.) | Властивості об'єкта ***Drive***. Сам об'єкт ***Drive*** створюється за допомогою методу ***GetDrive*** об'єкта ***FileSystemObject*** |
| Отримання відомостей про заданий каталог або файл (дата створення або останнього доступу, розмір, атрибути і т.д.) | Властивості об'єктів ***Folder*** і ***File***. Самі ці об'єкти створюються за допомогою методів ***GetFolder*** і ***GetFile*** об'єкта ***FileSystemObject*** |
| Перевірка існування певного диска, каталогу або файлу | Методи ***DriveExists***, ***FolderExists*** і ***FileExists*** об'єкта ***FileSystemObject*** |
| Копіювання файлів і каталогів | Методи ***CopyFile*** і ***CopyFolder*** об'єкта ***FileSystemObject***, а також методи ***File.Copy*** і ***Folder.Copy*** |
| Переміщення файлів і каталогів | Методи ***MoveFile*** і ***MoveFolder*** об'єкта ***FileSystemObject***, або методи ***File.Move*** і ***Folder.Move*** |
| Видалення файлів і каталогів | Методи ***DeleteFile*** і ***DeleteFolder*** об'єкта ***FileSystemObject***, або методи ***File.Delete*** і ***Folder.Delete*** |
| Створення каталогу | Методи ***FileSystemObject.CreateFolder*** або ***Folders.Add*** |
| Створення текстового файлу | Методи ***FileSystemObject.CreateTextFile*** або ***Folder.CreateTextFile*** |
| Отримання списку всіх доступних дисків | Колекція ***Drives***, що міститься у властивості ***FileSystemObject.Drives*** |
| Отримання списку всіх підкаталогів заданого каталогу | Колекція ***Folders***, що міститься у властивості ***Folder.SubFolders*** |
| Отримання списку всіх файлів заданого каталогу | Колекція ***Files***, що міститься у властивості ***Folder.Files*** |
| Відкриття текстового файлу для читання, запису або додавання | Методи ***FileSystemObject.CreateTextFile*** або ***File.OpenAsTextStream*** |
| Читання інформації з заданого текстового файлу або запис її в нього | Методи об'єкта ***TextStream*** |

## Об’єкт FileSystemObject

Об'єкт ***FileSystemObject*** є основним об'єктом, який забезпечує доступ до файлової системи комп'ютера, його методи використовуються для створення інших Об'єктів (***Drives***, ***Driv***e, ***Folders***, ***Folder***, ***Files***, ***File*** і ***Textstream***).

Для створення всередині сценарію об'єкта ***FileSystemObject*** можна скористатися методом ***CreateObject*** об'єкта ***WScript***:

***var fso = WScript.CreateObject("Scripting.FileSystemObject");***

Також можна використовувати об'єкт ***ActiveXObject*** мови ***JScript*** (за допомогою цього об'єкта можна працювати з файлової системою з сценаріїв, що знаходяться всередині HTML-сторінок):

***var fso = new ActiveXObject("Scripting.FileSystemObject");***

Об'єкт ***FileSystemObject*** має єдину властивість ***Drives***, в якому зберігається колекція, яка містить об'єкти ***Drive*** для всіх доступних дисків комп'ютера.

Методи об'єкта ***FiieSystemObject*** представлені в табл. 5.2.

Табл. 5.2. Методи об'єкта ***FileSystemObject***

|  |  |
| --- | --- |
| **Метод** | **Опис** |
| ***BuildPath(path, name)*** | Додає до заданого шляху (параметр ***path***) нове ім'я (параметр ***name***) |
| ***CopyFile(source, destination [, overwrite])*** | Копіює один або кілька файлів з одного місця (параметр ***source***) в інше (параметр ***destination***) |
| ***CopyFolder(source, destination [, overwrite])*** | Копіює каталог з усіма підкаталогами з одного місця (параметр ***source***) в інше (параметр ***destination***) |
| ***CreateFolder(foldername)*** | Створює новий каталог з ім'ям ***foldername***. Якщо каталог ***foldername*** вже існує, то станеться помилка |
| ***CreateTextFile(filename***  ***[, Overwrite [, Unicode]])*** | Створює новий текстовий файл з ім'ям ***filename*** і повертає об'єкт ***TextStream***, що вказує на цей файл |
| ***DeleteFile(filespec [, force])*** | Видаляє файл, шлях до якого заданий параметром ***filespec*** |
| ***DeleteFolder(folderspec [, force])*** | Видаляє каталог, шлях до якого заданий параметром ***folderspec***, разом з усім його вмістом |
| ***DriveExists(drivespec)*** | Повертає ***true***, якщо заданий параметром ***drivespec*** пристрій існує, і ***false*** - в протилежному випадку |
| ***FileExists(filespec)*** | Повертає ***true***, якщо заданий параметром ***filespec*** файл існує, і ***false*** - в протилежному випадку |
| ***FolderExists(folderspec)*** | Повертає ***true***, якщо заданий параметром folderspec каталог існує і ***false*** - в протилежному випадку |
| ***GetAbsolutePathName(partspec)*** | Повертає повний шлях для заданого відносного шляху ***pathspec*** (з поточного каталогу) |
| ***GetBaseName(path)*** | Повертає базове ім'я (без розширення) для останнього компонента у шляху ***path*** |
| ***GetDrive(drivespec)*** | Повертає об'єкт ***Drive***, що відповідає диску, заданому параметром ***drivespec*** |
| ***GetDriveName(path)*** | Повертає рядок, що містить ім'я диска в заданому шляху. Якщо з параметра ***path*** не можна виділити ім'я диска, то метод повертає порожній рядок (***""***) |
| ***GetExtensionName(path)*** | Повертає рядок, що містить розширення для останнього компонента у шляху ***path***. Якщо з параметра ***path*** не можна виділити компоненти шляху, то ***GetExtensionName*** повертає порожній рядок (***""***). Для мережевих дисків кореневої каталог (***\***) розглядається як компонент шляху |
| ***GetFile(filespec)*** | Повертає об'єкт ***File***, що відповідає файлу, заданому параметром ***filespec***. Якщо файл,, шлях до якого визначений у параметрі ***filespec***, не існує, то при виконанні методу виникне помилка |
| ***GetFileName(pathspec)*** | Повертає ім'я файлу, заданого повним шляхом до нього .Якщо з параметра ***pathspec*** можна виділити ім'я файлу, метод ***GetFileName*** повертає порожній  рядок (***""***) |
| ***GetFolder(folderspec)*** | Повертає об'єкт ***Folder***, що відповідає каталогу, заданому параметром ***folderspec***. Якщо каталог, шлях до якого визначеною у пункті ***folderspec***, не  існує, при виконанні методу виникне помилка |
| ***GetParentFolderName(path)*** | Повертає рядок, що містить ім'я батьківського каталогу для останнього компонента в заданому шляху. Якщо для останнього компонента в шляху, заданому параметром ***path***, не можна визначити батьківський каталог, то метод повертає порожній рядок (***""***) |
| ***GetSpecialFolder(folderpec)*** | Повертає об'єкт ***Folder*** для деяких спеціальних папок Windows, заданих числовим параметром ***folderspec*** |
| ***GetTempName()*** | Повертає випадковим чином сгенероване ім'я файлу або каталогу, яке може бути використане для операцій, що вимагають наявності тимчасового файлу або каталогу |
| ***MoveFile(source, destination)*** | Переміщує один або кілька файлів з одного місця (параметр ***source***) в інше (параметр ***destination***) |
| ***MoveFolder(source, destination)*** | Переміщує один або кілька каталогів з одного місця (параметр ***source***) в інше (параметр ***destination***) |
| ***OpenTextFile(filename***  ***(, iomode [, create [, format]]])*** | Відкриває заданий текстовий файл і повертає об'єкт ***TextStream*** для роботи з цим файлом |

**Об'єкт Drive**

За допомогою об'єкта ***Drive*** можна отримати доступ до властивостей заданого локального або мережевого диска. Створюється об'єкт ***Drive*** за допомогою методу ***GetDrive*** об'єкта ***FileSystemObject*** наступним чином:

***var fso, d;***

***fso = WScript.CreateObject("Scripting.FileSystemObject");***

***d = fso.GetDrive("C");***

Також об'єкти ***Drive*** можуть бути отримані як елементи колекції ***Drives***. Властивості об'єкта ***Drive*** представлені в табл 5.3, методів у цього об'єкта немає

Табл. 5.3. Властивості об'єкта ***Drive***

|  |  |
| --- | --- |
| **Властивість** | **Опис** |
| ***AvailableSpace*** | Містить кількість доступного для користувача місця (в байтах) на диску |
| ***DriveLetter*** | Містить букву, асоційовану з локальним пристроєм або мережевим ресурсом. Це властивість доступно 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, * 5 - RAM-диск |
| ***FileSystem*** | Містить тип файлової системи, що використовується на диску  (**FAT**, **NTFS** або **CDFS**) |
| ***FreeSpace*** | Містить кількість вільного місця (в байтах) на локальному диску або мережевому ресурсі. Доступно тільки для читання |
| ***IsReady*** | Містить ***true***, якщо пристрій готовий, і ***false*** - в протилежному випадку. Для пристроїв зі змінними носіями і приводів CD-ROM, повертає ***true*** тільки в тому випадку, коли в дисковод вставлений відповідний носій і пристрій готовий надати доступ до цього носія |
| ***Path*** | Містить шлях до диска (наприклад, ***C***, але не ***C\***) |
| ***RootFolder*** | Містить об'єкт ***Folder***, відповідний кореневому каталогу на диску. Доступно тільки для читання |
| ***SerialNumber*** | Містить десятковий серійний номер тому заданого диска |
| ***ShareName*** | Містить мережеве ім'я для диска. Якщо об'єкт не є мережевим диском, то у властивості ***ShareName*** міститься порожній рядок (***""***) |
| ***TotalSize*** | Містить загальний обсяг в байтах локального диска або мережевого ресурсу |
| ***VolumeName*** | Містить мітку тому для диска. Доступно для читання і запису |

**Колекція Drives**

Доступна тільки для читання колекція ***Drives*** містить об'єкти ***Drive*** для всіх доступних дисків комп'ютера, в тому числі для мережевих дисків і дисководів зі змінними носіями.

У властивості ***count*** колекції ***Drives*** зберігається число її елементів, тобто число доступних дисків.

***var FSO, DriveCol, d;***

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

***DriveCol = fso.Drives;***

***//Витягнення елемента колекції***

***d = DriveCol.Item("С:");***

***WScript.Echo(“Disk С has ",d.FreeSpace,"kb free");***

Для перебору всіх елементів колекції ***Drives*** потрібно використовувати об'єкт ***Enumerator***, який був описаний у попередній лекції, де також був приведений приклад сценарію виведення інформації про всі диски.

**Об'єкт Folder**

Об'єкт ***Folder*** забезпечує доступ до властивостей каталогу. Створити цей об'єкт можна за допомогою властивості ***RootFolder*** об'єкта ***Drive*** або методів ***GetFolder***, ***GetParentFolder*** і ***GetSpecialFolder*** об'єкта ***FileSystemObject*** наступним чином:

***var fso, folder;***

***fso = WScript.CreateObject("Scripting.FileSystemObject");***

***folder = fso.GetFolder ("С:\\My Documents");***

Також об'єкти ***Folder*** можуть бути отримані як елементи колекції ***Folders***.

Властивості об'єкта ***Folder*** представлені в табл. 5.4.

Табл. 5.4. Властивості об'єкта ***Folder***.

|  |  |
| --- | --- |
| **Властивість** | **Опис** |
| ***Attributes*** | Дозволяє переглянути або встановити атрибути каталогу |
| ***DateCreated*** | Містить дату та час створення каталогу. Доступно тільки для читання |
| ***DateLastAccessed*** | Містить дату та час останнього доступу до каталогу. Доступно тільки для читання |
| ***DateLastModified*** | Містить дату та час останньої модифікації каталогу. Доступно тільки для читання |
| ***Drive*** | Містить необхідну букву пристрою, на якому знаходиться каталог. Доступно тільки для читання |
| ***Files*** | Містить колекцію ***Files***, що складається з об'єктів ***File*** для всіх файлів в каталозі (включаючи приховані і системні) |
| ***IsRootFolder*** | Містить ***true***, якщо каталог є кореневим, і ***false*** - в іншому випадку |
| ***Name*** | Дозволяє переглянути і змінити ім'я каталогу Доступно для читання і запису |
| ***ParentFolder*** | Містить об'єкт ***Folder*** для батьківського каталогу. Досупная тільки для читання |
| ***Path*** | Містить шлях до каталогу |
| ***ShortName*** | Містить коротку назву каталогу |
| ***ShortPath*** | Містить шлях до каталогу, що складається з коротких імен каталогів |
| ***Size*** | Містить розмір всіх файлів і підкаталогів, що входять в даний каталог, в байтах |
| ***SubFolders*** | Містить колекцію Folders, що складається з усіх підкаталогів каталогу (включаючи підкаталоги з атрибутами "Прихований" і "Системний") |
| ***Type*** | Містить інформацію про тип каталогу |

Методи об'єкта ***Folder*** описані в табл. 5.5.

Табл. 5.5. Методи об'єкта ***Folder***

|  |  |
| --- | --- |
| **Метод** | **Опис** |
| ***Copy(destination [, overwrite])*** | Копіює каталог в інше місце |
| ***CreateTextFile(filename [,overwrite [,Unicode]])*** | Створює новий текстовий файл з ім'ям ***filename*** І повертає об'єкт TextStream, що вказує на цей файл (цей метод аналогічний розглянутому вище методу ***CreateTextFile*** об'єкта ***FileSystemObject***) |
| ***Delete([force])*** | Видаляє каталог |
| ***Move(destination)*** | Переміщує каталог в інше місце |

**Колекція Folders**

Колекція ***Folders*** містить об'єкти ***Folder*** для всіх підкаталогів видаленного каталогу. Створюється ця колекція з допомогою властивості ***SubFolders*** відповідного об'єкта ***Folder***. Наприклад, в наступному прикладі змінна ***fc*** є колекцією, що містить об'єкти ***Folder*** для всіх підкаталогів каталогу ***C:\Program Files***.

***var fso, f, fc;***

***fso = WScnpt.CreateObject ("Scripting FileSystemObject");***

***f = fso.GetFolder("C:\\Program Files");***

***fc = f .Sub Folders;***

Колекція ***Folders*** (як і ***Drives***) має властивість ***Count*** та метод ***Item***. Крім цього, у ***Folders*** є метод ***Add(folderName)***, дозволяє створювати нові підкаталоги.

Для доступу до всіх елементів колекції, потрібно використовувати, як звичайно об'єкт ***Enumerator***.

***var fso, f, fc, s;***

***fso = WScript.CreateObject("Scripting.FileSystemObject");***

***f = fso.GetFolder("C:\\ProgramFiles");***

***fc = new Enumerator(f.SubFolders);***

***s = "";***

***for (; 'fс.atEnd(); fc.moveNext()) {***

***s += fc.item();***

***s += "\n";***

***}***

***WScript.Echo(s);***

**Об'єкт File**

Об'єкт ***File*** забезпечує доступ до всіх властивостей файлу. Створити цей об'єкт можна за допомогою методу ***GetFile*** об'єкта ***FileSystemObject*** наступним чином:

***var fso, f;***

***fso = WScript.CreateObject ("Scripting.FileSystemObject");***

***f = fso.GetFile("C:\\My Documents\\letter.txt");***

Також об'єкти ***File*** можуть бути отримані як елементи колекції ***Files***. Властивості об'єкта ***File*** описані в табл 5.6.

Табл. 5.6. Властивості об'єкта ***File***

|  |  |
| --- | --- |
| **Властивість** | **Опис** |
| ***Attributes*** | Дозволяє переглянути або встановити атрибути файлів |
| ***DateCreated*** | Містить дату та час створення файлу. Доступно тільки для читання |
| ***DateLastAccessed*** | Містить дату та час останнього доступу до файлу. Доступно тільки для читання |
| ***DateLastModffied*** | Містить дату та час останньої модифікації файлу. Доступно тільки для читання |
| ***Drive*** | Містить букву диска для пристрою, на якому знаходиться файл. Доступно тільки для читання |
| ***Name*** | Дозволяє переглянути і змінити ім'я файлу. Доступно для читання і запису |
| ***ParentFolder*** | Містить об'єкт ***Folder*** для батьківського каталогу файла. Доступно 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*** | Містить розмір заданого файлу в байтах |
| ***Туре*** | Повертає інформацію про тип файлу. Наприклад, для файлу з розширенням ***txt*** повернеться рядок ***Text Document*** |

Методи об'єкта ***File*** представлені в табл. 5.7.

Табл. 5.7. Методи об'єкта ***File***

|  |  |
| --- | --- |
| **Метод** | **Опис** |
| ***Copy(destination [, overwrite])*** | Копіює файл в інше місце |
| ***Delete([force])*** | Видаляє файл |
| ***Move(destination)*** | Переміщує файл в інше місце |
| ***OpenAsTextStream([iomode, [format]])*** | Відкриває заданий файл і повертає об'єкт ***TextStream***, який може бути використаний для читання, запису або додавання даних в текстовий файл |

**Колекція Files**

Колекція ***Files*** містить об'єкти File для всіх файлів, що знаходяться всередині певного каталогу. Створюється ця колекція з допомогою властивості ***Files*** відповідного об'єкта ***Folder***. Для доступу в циклі до всіх елементів колекції ***Files*** використовується об'єкт ***Enumerator***.

***var fso, f, fl, fc, s;***

***fso = WScript.CreateObject("Scripting.FileSystemObject");***

***f = fso.GetFolder("С:\\My Documents");***

***fc = new Enumerator(f.files),***

***for (; !fс.atEnd(); fс.moveNext())***

***{***

***s += fc.item();***

***s += "\n";***

***}***

***WScript.Echo(s);***

## Приклади сценаріїв

Далі наведені прості приклади сценаріїв, які працюють з файлової системою (створення, копіювання, видалення файлів і каталогів, читання і запис рядків у текстовому файлі і т. д.).

### Отримання відомостей про диск

Доступ до властивостей заданого локального або мережевого диска можна отримати за допомогою об'єкта ***Drive***, який повертається методом ***GetDrive*** об'єкта ***FileSystemObject***, а також може бути отриманий як елемент колекції ***Drives***.

Далі приведений сценарій ***DriveInfo.js***, який виводить на екран деякі властивості диска ***С***.

***//Оголошуємо змінні***

***var FSO, d, TotalSize, FreeSpace, s;***

***//Створюємо об'єкт FileSystemObject***

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

***//Створюємо об'єкт Drive для диска C***

***d = FSO.GetDrive("C:");***

***s = "Information about drive C:\n";***

***//Отримуємо серійний номер диска***

***s += "Serial number: " + d.SerialNumber + "\n";***

***//Отримуємо мітку тому диска***

***s += "Label: " + d.VolumeName + "\n";***

***//Обчислюємо загальний обсяг диска в кілобайтах***

***TotalSize = d.TotalSize/1024;***

***s += "Total: " + TotalSize + " Kb\n";***

***//Обчислюємо обсяг вільного простору диска в кілобайтах***

***FreeSpace = d.FreeSpace/1024;***

***s += "Free: " + FreeSpace + " Kb\n";***

***//Виводимо властивості диска на екран***

***WScript.Echo(s);***

### Отримання відомостей про каталог

Доступ до властивостей каталогу забезпечує об'єкт ***Folder***. Створити цей об'єкт можна за допомогою властивості ***RootFolder*** об'єкта ***Drive*** або методів ***GetFolder***, ***GetParentFolder*** і ***GetSpecialFolder*** об'єкта ***FileSystemObject***. Також об'єкти ***Folder*** можуть бути отримані як елементи колекції ***Folders***.

У сценарії ***FolderInfo.js*** на екран виводяться властивості каталогу, з якого був запущений сценарій.

***var FSO, WshShell, FoldSize, s; //Оголошуємо змінні***

***//Створюємо об'єкт FileSystemObject***

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

***//Створюємо об'єкт WshShell***

***WshShell = WScript.CreateObject("WScript.Shell");***

***//Визначаємо каталог, з якого був запущений сценарій***

***//(Поточний каталог)***

***Folder = FSO.GetFolder(WshShell.CurrentDirectory);***

***//Отримуємо ім'я поточного каталогу***

***s = "Current directory: " + Folder.Name + "\n";***

***//Отримуємо дату створення поточного каталогу***

***s += "Creation date: " + Folder.DateCreated + "\n";***

***//Обчислюємо розмір поточного каталогу в кілобайтах***

***FoldSize = Folder.Size/1024;***

***s += "Total size: " + FoldSize + " Kb\n";***

***//Виводимо інформацію на екран***

***WScript.Echo(s);***

### Отримання інформації про файл

Доступ до всіх властивостей файлу забезпечує об'єкт ***File***, створити який можна за допомогою колекції ***Files*** або методу ***GetFile*** об'єкта ***FileSystemObject***.

Далі приведений сценарій ***FileInfo.js***, в якому на екран виводяться деякі властивості файлу ***C:\mtsearchdump.txt***.

***var FSO, f, s //Оголошуємо змінні***

***//Створюємо об'єкт FileSystemObject***

***FSO = WScript.CreateObject("Scripting.FileSystemObject")***

***//Створюємо об'єкт File***

***f = FSO.GetFile("C:\\mtsearchdump.txt")***

***//Отримуємо ім'я файлу***

***s = "File: " + f.Name + "\n";***

***//Отримуємо дату створення файлу***

***s += "Creation date: " + f.DateCreated + "\n";***

***//Отримуємо тип файлу***

***s += "Type: " + f.Type + "\n";***

***//Виводимо інформацію на екран***

***WScript.Echo(s);***

### Створення каталогу

Створити новий каталог на диску можна або за допомогою методу ***CreateFolder*** об'єкта ***FileSystemObject***, або за допомогою методу ***Add*** колекції ***Folders***. Обидва ці методи використовуються в сценарії ***MakeFolder.js*** для створення на диску ***N*** каталогу ***New folder*** та підкаталогу ***Another folder*** у ньому.

***//Оголошуємо змінні***

***var FSO, f, SubFolders;***

***//Створюємо об'єкт FileSystemObject***

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

***//Створюємо каталог N:\NewFolder***

***FSO.CreateFolder("N:\\New folder");***

***//Створюємо об'єкт Folder для каталогу N:\NewFolder***

***f = FSO.GetFolder("N:\\New folder");***

***//Створюємо колекцію підкаталогів каталогу N:\NewFolder***

***SubFolders = f.SubFolders;***

***//Створюємо каталог N:\New folder\Another folder***

***SubFolders.Add("Another folder");***

**Створення текстового файлу**

Для створення текстового файлу використовується метод ***CreateTextFile*** об'єкта ***FileSystemObject***, який має один обов'язковий текстовий параметр (шлях до створюваного файлу) і два необов'язкових логічних параметра (***Overwrite*** і ***Unicode***).

Параметр ***Overwrite*** має значення в тому випадку, коли створюється файл вже існує. Якщо ***Overwrite*** дорівнює ***True***, то такий файл перепишеться (старий вміст буде втрачений), якщо ж в якості ***Overwrite*** вказано ***False***, то файл перезаписаний не буде. Якщо цей параметр взагалі не вказано, то існуючий файл також не буде переписаний.

Параметр Unicode вказує, в якому форматі (ASCII або Unicode) слід створювати файл. Якщо Unicode одно True, то файл створюється в форматі Unicode, якщо ж Unicode одно False або цей параметр взагалі не вказано, то файл створюється в режимі ASCII.

У сценарії ***CreateTempFile.js*** показано, яким чином можна створити файл зі випадково обраним ім'ям (такі файли часто використовуються для запису тимчасових даних).

***var FSO, FileName, f, s; //Оголошуємо змінні***

***//Створюємо об'єкт FileSystemObject***

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

***//Генеруємо випадкове ім'я файлу***

***FileName = FSO.GetTempName();***

***//Створюємо файл з ім'ям FileName***

***f = FSO.CreateTextFile(FileName, true);***

***//Закриваємо файл***

***f.Close();***

***//Повідомляємо про створення файлу***

***WScript.Echo("File " + FileName + " is created");***

### Копіювання і переміщення файлів і каталогів

Для копіювання файлів/каталогів можна застосовувати метод ***CopyFile/CopyFolder*** об'єкта ***FileSystemObject*** або метод ***Copy*** відповідного цього файлу/каталогу об'єкта ***File/Folder***. Переміщаються файли/каталоги за допомогою методів ***MoveFile/MoveFolder*** об'єкта ***FileSystemObject*** або методу ***Move*** відповідного цього файлу/каталогу об'єкта ***File/Folder***.

Відзначимо, що при використанні всіх цих методів процес копіювання або переміщення переривається після першої помилки, що виникла. Крім того, не можна переміщати файли і каталоги з одного диска на інший.

Далі наведено сценарій ***CopyFile.js***, який ілюструє використання методу ***Copy***. У цьому сценарії на диску ***C*** створюється файл ***TestFile.txt***, який потім копіюється на робочий стіл.

***//Оголошуємо змінні***

***var FSO, f, WshShell, WshFldrs, PathCopy;***

***//Створюємо об'єкт FileSystemObject***

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

***//Створюємо файл***

***f = FSO.CreateTextFile("N:\\TestFile.txt",true);***

***//Записуємо в файл рядок***

***f.WriteLine("Test file");***

***//Закриваємо файл***

***f.Close();***

***//Створюємо об'єкт WshShell***

***WshShell = WScript.CreateObject("WScript.Shell");***

***//Створюємо об'єкт WshSpecialFolders***

***WshFldrs = WshShell.SpecialFolders;***

***//Визначаємо шлях до робочого столу***

***PathCopy = WshFldrs.item("Desktop") + "\\";***

***//Створюємо об'єкт File для файлу N:\TestFile.txt***

***f = FSO.GetFile("N:\\TestFile.txt");***

***//Копіюємо файл на робочий стіл***

***f.Copy(PathCopy);***

### Видалення файлів і каталогів

Для видалення файлів/каталогів можна застосовувати метод ***DeleteFile/DeleteFolder*** об'єкта ***FileSystemObject*** або метод ***Delete*** відповідного цього файлу/каталогу об'єкта ***File/Folder***. Відзначимо, що при видаленні каталогу неважливо, чи є він порожнім чи ні - видалення буде вироблено в будь-якому випадку. Якщо ж заданий для видалення файл/каталог не буде знайдений, то виникне помилка.

Далі наведено сценарій ***DeleteFile.js***, в якому проводиться видалення попередньо створеного файлу ***C:\TestFile.txt***.

***var FSO, f, FileName;***

***//Створюємо об'єкт FileSystemObject***

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

***//Задаємо ім'я файлу***

***FileName = "N:\\TestFile.txt";***

***//Створюємо файл***

***f = FSO.CreateTextFile(FileName, true);***

***//Записуємо в файл рядок***

***f.WriteLine("Test file");***

***//Закриваємо файл***

***f.Close();***

***WScript.Echo("File is created");***

***FSO.DeleteFile(FileName);***

***WScript.Echo("File is deleted");***

**Лекція №6. «Об’єктна модель WSH: основні об’єкти, їх властивості та методи»**

## Об'єкт WshShell

За допомогою об'єкта ***WshShell*** можна запускати новий процес, створювати ярлики, працювати з системним реєстром, отримувати доступ до змінних середовища і спеціальним папок Windows. Створюється цей об'єкт наступним чином:

***var WshShell = WScript.CreateObject("WScript.Shell");***

Об'єкт ***WshShell*** має три властивості, які наведені у табл. 6.1.

Табл. 6.1. Властивості об’єкта ***WshShell***

|  |  |
| --- | --- |
| **Властивість** | **Опис** |
| ***CurrentDirectory*** | Тут зберігається повний шлях до поточного каталогу (до каталогу, з якого був запущений сценарій) |
| ***Environment*** | Містить об'єкт ***WshEnvironment***, який забезпечує доступ до змінних середовища операційної системи для Windows NT/2000 /XP або до змінних середовища поточного командного вікна для Windows 9x |
| ***SpecialFolders*** | Містить об'єкт ***WshSpecialFolders*** для доступу до спеціальних папках Windows (робочий стіл, меню Пуск (Start) і т. Д.) |

Наведемо тепер методи, наявні в об'єкта ***WshShell***.

Табл. 6.2. Методи об’єкта ***WshShell***

|  |  |
| --- | --- |
| **Метод** | **Опис** |
| ***AppActivate(title)*** | Активізує заданий параметром title вікно програми. Рядок title задає назву вікна (наприклад, "calc" або "notepad") або ідентифікатор процесу (Process ID, PID) |
| ***CreateShortcut(strPathname)*** | Створює об'єкт ***WshShortcut*** для зв'язку з ярликом Windows (розширення lnk) або об'єкт ***WshUrlShortcut*** для зв'язку з мережевим ярликом (розширення url). Параметр ***strPathname*** задає повний шлях до створюваного або змінюваному ярлику |
| ***Environment(strType)*** | Повертає об'єкт ***WshEnvironment***, в якому знаходяться середовища заданого виду |
| ***Exec(strCommand)*** | Створює новий дочірній процес, який запускає консольний додаток, заданий параметром ***strCommand***. В результаті повертається об'єкт ***WshScriptExec***, що дозволяє контролювати хід виконання запущеного додатку і забезпечує доступ до водних потоків ***StdIn***, ***StdOut*** і ***StdErr*** цього додатка |
| ***ExpandEnvironmentStrings(strString)*** | Повертає значення змінної середовища поточного командного вікна, заданим рядком ***strString*** (ім'я змінної повинно бути оточене знаками "%") |
| ***LogEvent(intType,strMessage[, strTarget])*** | Протоколює події в журналі Windows або в файлі WSH.log. Цілочисельний параметр ***intType*** визначає тип повідомлення, рядок ***strMessage*** - текст повідомлення. Параметр ***strTarget*** він визначає назву системи, в якій записуються події (за замовчуванням це локальна система). Метод ***LogEvent*** повертає ***true***, якщо подія записано успішно і ***false*** в іншому випадку |
| ***Popup(strText, [nSecToWait], [strTitle], [nType])*** | Виводить на екран інформаційне вікно з повідомленням, заданим параметром ***strText***. Параметр ***nSecToWait*** задає кількість секунд, після закінчення яких вікно буде автоматично закрито, параметр ***strTitle*** визначає заголовок вікна, параметр ***nType*** вказує тип кнопок і значка для вікна |
| ***RegDelete(strName)*** | Видаляє з системного реєстру заданий параметр або розділ цілком |
| ***RegRead(strName)*** | Повертає значення параметра реєстру або значення за замовчуванням для розділу реєстру |
| ***RegWrite(strName, anyValue [, strType])*** | Записує до реєстру значення заданого параметра або значення за замовчуванням для розділу |
| ***Run(strCommand, [intWindowStyle], [bWaitOnReturn])*** | Створює новий незалежний процес, який запускає додаток, заданий параметром strCommand |
| ***SendKeys(string)*** | Посилає одне або кілька натискань клавіш в активне вікно (ефект той же, як якщо б ви натискали ці клавіші на клавіатурі) |
| ***SpecialFolders(strSpecFolder)*** | Повертає рядок, що містить шлях до спеціальній папці Windows, заданої параметром ***strSpecFolder*** |

Розглянемо більш докладно деякі методи, наведені в таблиці.

**Метод AppActivate**

Метод AppActivate активує вже запущений вказаний додаток (встановлює на нього фокус), але не виробляє ніяких дій зі зміни розмірів його вікна. Для того щоб спочатку запустити потрібну програму і визначити вид його вікна, треба використовувати метод ***Run*** об'єкта ***WshShell***. Для того щоб визначити, який самий додаток необхідно активізувати, рядок title порівнюється по черзі з назвами вікон всіх запущених додатків. Якщо не знайдено жодного точного збігу, буде проводитись пошук того додатка, назва вікна якого починається з рядка ***title***. Якщо і в цьому випадку не буде знайдено жодного підходящого додатка, то буде вестися пошук додатка, заголовок якого закінчується на цей рядок. Якщо буде буде знайдено декілька відповідних вікон, то відбудеться активізація одного з них (вікно обирається довільно).

**Метод CreateShortcut**

Цей метод дозволяє створити новий або відкрити вже існуючий ярлик для зміни його властивостей.

Наведемо приклад сценарію, в якому створюються два ярлика - на сам виконуваний сценарій (об'єкт ***oShellLink***) і на мережевий ресурс (***oUrlLink***).

***var WshShell, oShellLink, oUrlLink;***

***//Створюємо об'єкт WshShell***

***WshShell = WScript.CreateObject("WScript.Shell");***

***//Створюємо ярлик на файл***

***oShellLink = WshShell.CreateShortcut("Current Script.lnk");***

***//Встановлюємо шлях до файлу***

***oShellLink.TargetPath = WScript.ScriptFullName;***

***//Зберігаємо ярлик***

***oShellLink.Save();***

***//Створюємо ярлик на мережевий ресурс***

***oUrlLink = WshShell.CreateShortcut("Microsoft Web Site.URL");***

***//Встановлюємо URL***

***oUrlLink.TargetPath = "http://www.microsoft.com";***

***//Зберігаємо ярлик***

***oUrlLink.Save();***

**Метод Popup**

Якщо в методі не заданий параметр ***strTitle***, то заголовком вікна буде "Windows Script Host."

Параметр ***nType*** може приймати ті ж значення, що і в функції ***MessageBox*** з Microsoft Win32 API. У табл. 6.3 описані деякі можливі значення параметра ***nType*** і їх зміст.

Табл. 6.3. Типи кнопок та іконок для методу ***Popup***

|  |  |
| --- | --- |
| **Значення nТуре** | **Опис** |
| ***0*** | Виводиться кнопка **Ok** |
| ***1*** | Виводятьcя кнопки **Ok** і **Cancel** |
| ***2*** | Виводяться кнопки **Abort**, **Retry** і **Ignore** |
| ***3*** | Виводяться кнопки **Yes**, **No** і **Cancel** |
| ***4*** | Виводяться кнопки **Yes** і **No** |
| ***5*** | Виводяться кнопки **Retry** і **Cancel** |
| ***16*** | Виводиться іконка **Stop** **Mark** |
| ***32*** | Виводиться іконка **Question Mark** |
| ***48*** | Виводиться іконка **Exclamation Mark** |
| ***64*** | Виводиться іконка **Information Mark** |

Природно, в методі ***Popup*** можна комбінувати значення параметра, наведені в табл. 6.3.

Метод ***Popup*** повертає ціле значення, за допомогою якого можна дізнатись, яка саме кнопка була натиснута для виходу (табл. 6.4).

Табл. 6.4. Значення, що повертаються методом ***Popup***

|  |  |
| --- | --- |
| **Значення** | **Опис** |
| ***-1*** | Користувач не натиснув на жодну з кнопок протягом часу, заданого параметром ***nSecToWait*** |
| ***1*** | Натиснута кнопка **Ok** |
| ***2*** | Натиснута кнопка **Cancel** |
| ***3*** | Натиснута кнопка **Abort** |
| ***4*** | Натиснута кнопка **Retry** |
| ***5*** | Натиснута кнопка **Ignore** |
| ***6*** | Натиснута кнопка **Yes** |
| ***7*** | Натиснута кнопка **No** |

**Метод Run**

Параметр ***intWindowStyle*** встановлює вид вікна для додатку, що запускається (табл. 6.5).

Табл. 6.5. Типи вікна (***intWindowStyle***)

|  |  |
| --- | --- |
| **Параметр** | **Опис** |
| ***0*** | Приховує поточне вікно і активує інше вікно |
| ***1*** | Активує і відображує вікно. Якщо вікно було мінімізовано або максимізовано, система відновить його початкове розташування і розмір. Цей прапорець повинен вказуватися сценарієм під час першого відображення вікна |
| ***2*** | Активує вікно і відображає його в мінімізованому (згорнутому) вигляді |
| ***3*** | Активує вікно і відображає його в максимізувати (розгорнутому) вигляді |
| ***4*** | Показує вікно в тому вигляді, в якому воно знаходилося останній раз. Активне вікно при цьому залишається активним |
| ***5*** | Активує вікно і відображує його в поточному стані |
| ***6*** | Мінімізує задане вікно і активує наступне (в Z-порядку) вікно |
| ***7*** | Показує вікно в згорнутому вигляді. Активне вікно при цьому залишається активним |
| ***8*** | Показує вікно в його поточному стані. Активне вікно при цьому залишається активним |
| ***9*** | Активує і відображає вікно. Якщо вікно було мінімізовано або максимізувало, система відновить його початкове розташування і розмір. Цей прапорець повинен вказуватися сценарієм, якщо виводиться відновлення згорнутого вікна |
| ***10*** | Встановлює режим відображення, що спирається на режим програми, яка запускає додаток |

Необов'язковий параметр ***bWaitOnReturn*** є логічною змінною, дає вказівку очікувати завершення запущеного процесу. Якщо цей параметр не зазначений або встановлений в ***false***, то після запуску з сценарію нового процесу управління відразу ж повертається назад в сценарій (не чекаючи завершення запущеного процесу). Якщо ж ***bWaitOnReturn*** встановлений в ***true***, то сценарій відновить роботу тільки після завершення викликаного процесу.

При цьому якщо параметр ***bWaitOnReturn*** дорівнює ***true***, то метод ***Run*** повертає код виходу викликаного додатку. Якщо ж ***bWaitOnReturn*** дорівнює ***false*** або не заданий, то метод ***Run*** завжди повертає нуль.

У наступному прикладі ми запускаємо Microsoft Notepad і відкриваємо в ньому файл з виконуваним сценарієм:

***var WshShell = WScript.CreateOb]ect("WScript.Shell");***

***WshShell.Run("%windir%\\notepad " + WScript.ScriptFullName);***

Наступний сценарій друкує код виходу викликаного програми:

***var WshShell = WScript.CreateObject("WScript.Shell");***

***Return = WshShell.Run("notepad " + WScript.ScriptFullName, 1, true);***

***WScript.Echo("Код возврата:", Return);***

**Метод Send Keys**

Кожна клавіша задається одним або декількома символами. Наприклад для того щоб задати натискання один за одним букв ***А***, ***Б*** і ***В***, потрібно вказати в якості параметра для ***SendKeys*** рядок "***АБВ***": ***string = "АБВ "***.

Кілька символів мають в методі ***SendKeys*** спеціальне значення: ***+***, ***^***, ***%***, ***~***, ***(***, ***)***. Для того щоб задати один з цих символів, їх потрібно укласти в фігурні дужки (***{}***). Наприклад, для завдання знака плюс використовується ***{+}***.

Квадратні дужки (***[]***) хоча і не мають в методі ***SendKeys*** спеціального сенсу, їх також потрібно укладати в фігурні дужки. Крім цього, для завдання самих фігурних дужок слід використовувати такі конструкції: ***{{}*** (ліва дужка) і ***{}}*** (права дужка).

Для завдання символів, що не відображаються, таких як ***<Enter>*** або ***<Tab>***, і спеціальних клавіш в методі ***SendKeys*** використовуються коди, представлені в табл. 6.6.

Табл. 6.6. Коди спеціальних клавіш для ***SendKeys***

|  |  |  |  |
| --- | --- | --- | --- |
| **Клавіша** | **Код** | **Клавіша** | **Код** |
| **<Backspace>** | ***{BACKSPACE}, {BS}*** або ***{BKSP}*** | **<Стрілка вниз>** | ***{DOWN}*** |
| **<Break>** | ***{BREAK}*** | **<Стрілка вправо>** | ***{RIGHT}*** |
| **<Caps Lock>** | ***{CAPSLOCK}*** | **<F1>** | ***{F1}*** |
| **<Del>** або **<Delete>** | ***{DELETE}*** або ***{DEL}*** | **<F2>** | ***{F2}*** |
| **<End>** | ***{END}*** | **<F3>** | ***{F3}*** |
| **<Enter>** | ***{ENTER}*** або ***~*** | **<F4>** | ***{F4}*** |
| **<Esc>** | ***{ESC}*** | **<F5>** | ***{F5}*** |
| **<Help>** | ***{HELP}*** | **<F6>** | ***{F6}*** |
| **<Home>** | ***{HOME}*** | **<F7>** | ***{F7}*** |
| **<Ins>** або**<Insert>** | ***{INSERT}*** або ***{INS}*** | **<F8>** | ***{F8}*** |
| **<Num Lock>** | ***{NUMLOCK}*** | **<F9>** | ***{F9}*** |
| **<Page Down>** | ***{PGDN}*** | **<F10>** | ***{F10}*** |
| **<Page Up>** | ***{PGUP}*** | **<F11>** | ***{F11}*** |
| **<Print Screen>** | ***{PRTSC}*** | **<F12>** | ***{F12}*** |
| **<Scroll Lock>** | ***{SCROLLLOCK}*** | **<F13>** | ***{F13}*** |
| **<Tab>** | ***{TAB}*** | **<F14>** | ***{F14}*** |
| **<Стрілка вгору>** | ***{UP}*** | **<F15>** | ***{F15}*** |
| **<Стрілка вліво>** | ***{LEFT}*** | **<F16>** | ***{F16}*** |

Для завдання комбінацій клавіш з ***<Shift>***, ***<Ctrl>*** або ***<Alt>*** перед відповідною клавішею потрібно поставити один або кілька кодів з табл. 6.7.

Табл. 6.7. Коди клавіш для ***<Shift>***, ***<Ctrl>*** та ***<Alt>***

|  |  |
| --- | --- |
| **Клавіша** | **Код** |
| **<Shift>** | ***+*** |
| **<Ctrl>** | ***^*** |
| **<Alt>** | ***%*** |

Для того щоб задати комбінацію клавіш, яку потрібно набирати, утримуючи комбінацію клавіш ***<Shift>***, ***<Ctrl>*** або ***<Alt>***, необхідно укласти коди цих клавіш в дужки. Наприклад, якщо потрібно зімітувати натиснення клавіш ***<G>*** і ***<S>***, утримуючи клавішу ***<Shift>***, слід використовувати послідовність ***+(GS)***. Для того ж, щоб задати одночасне натиснення клавіш ***<Shift>*** і ***<G>***, а потім ***<S>*** (вже без ***<Shift>***), використовується ***+GS***.

У методі ***SendKeys*** можна задати кілька натискань поспіль однієї і тієї ж клавіші. Для цього необхідно в фігурних дужках вказати код потрібної клавіші, а через пробіл - число натиснень. Наприклад, ***{LEFT 42}*** означає натискання клавіші **<Стрілка вліво>** 42 рази поспіль; ***{H 10}*** означає натискання клавіші **<h>** 10 разів поспіль.

**Об'єкт WshShortcut**

За допомогою об'єкта ***WshShortcut*** можна створити новий ярлик Windows або змінити властивості вже існуючого ярлика Даний об'єкт можна створити тільки за допомогою методу ***CreateShortcut*** об'єкта WshShell. У наступному прикладі зі сценарію створюється ярлик на цей самий сценарій (ярлик буде знаходитися в поточному каталозі):

***var WshShell = WScript.CreateObject("WScript.Shell");***

***var oShellLink = WshShell.CreateShortcut("Current Script.Ink");***

***oShellLink.TargetPath = WScript.ScnptFullName;***

***oShellLink.Save(};***

Властивості об'єкта ***WshShortcut*** описані в табл. 6.8.

Табл. 6.8. Властивості об'єкта ***WshShortcut***

|  |  |
| --- | --- |
| **Властивість** | **Опис** |
| ***Arguments*** | Містить рядок, що задає параметри командного рядка для ярлика |
| ***Description*** | Містить опис ярлика |
| ***FullName*** | Містить рядок з повним шляхом до ярлику |
| ***HotKey*** | Задає гарячу клавішу для ярлика, тобто визначає комбинацію клавіш, за допомогою яких можна запустити або зробити активною програму, на яку вказує заданий ярлик |
| ***IconLocation*** | Задає шлях до іконки ярлика |
| ***TargetPath*** | Встановлює шлях до файлу, на який вказує ярлик |
| ***WindowStyle*** | Визначає вигляд вікна для додатка, на яке вказує ярлик |
| ***WorkingDirectory*** | Задає робочий каталог для додатка, на яке вказує ярлик |

У наступному прикладі створюється ярлик для поточного сценарію з двома параметрами:

***var WshShell = WScript.CreateOb]ect("WScript.Shell");***

***var oShellLink = WshShell.CreateShortcut("Current Script.Ink");***

***oShellLmk.TargetPath = WScript. ScnptFullName;***

***oShellLink.Arguments = "-a abc.txt";***

***oShellLink.Save();***

Для того щоб призначити ярлику гарячу клавішу, необхідно у властивість ***HotKey*** записати рядок, що містить назви потрібних клавіш, розділені символом "***+***".

У наступному прикладі на робочому столі створюється ярлик для Блокнота, якому призначається гаряча клавіша (комбінація **<Ctrl>+<Alt>+<D>**):

***var WshShell = WScript.CreateOrrject("WScript .Shell") ;***

***strDesktop = WshShell.SpecialFolders("Desktop") ;***

***oMyShortcut = WshShell.CreateShortcut(strDesktop+"\\a\_key. Ink") ;***

***oMyShortcut.TargetPath = "%wmdir%\\notepad.exe";***

***oMyShortcut.Hotkey = "CTRL+ALT+D";***

***oMyShortcut.Save() ;***

***WScript.Echo(oMyShortcut.Hotkey);***

## Об'єкти-колекції

В WSH входять об'єкти, за допомогою яких можна отримати доступ до колекцій, що містить наступні елементи:

* параметри командного рядка запущеного сценарію або ярлика Windows (об'єкти ***WshArguments***, ***WshNamed*** і ***WshUnnamed***);
* значення змінних середовища (об'єкт ***WshEnvironment***);
* шляху до спеціальних папках Windows (об'єкт ***WshSpecialFolders***).

### Об'єкт WshArguments

Об'єкт ***WshArguments*** містить колекцію всіх параметрів командного рядка запущеного сценарію або ярлика Windows. Цей об'єкт можна створити тільки за допомогою властивості ***Arguments*** об'єктів ***WScript*** і ***WshShortcut***.

За допомогою об'єкта ***WshArguments*** можна також виділяти і окремо обробляти аргументи сценарію, у яких є імена (наприклад, ***/Name:Andrey***) і безіменні аргументи. Ясно, що використання іменних параметрів зручніше, так як в цьому випадку немає необхідності запам'ятовувати, в якому порядку повинні бути записані параметри при запуску того чи іншого сценарію.

Для доступу до іменних і безіменним аргументів використовуються відповідно два спеціальних властивості об'єкта ***WshArguments***: ***Named*** і ***Unnamed***. Властивість ***Named*** містить посилання на колекцію ***WshNamed***, властивість ***Unnamed*** - на колекцію ***WshUnnamed***.

Таким чином, обробляти параметри командного рядка запущеного сценарію можна трьома способами:

* переглядати повний набір всіх параметрів (як іменних, так і безіменних) за допомогою колекції ***WshArguments***;
* виділити тільки ті параметри, які мають імена (іменні параметри) за допомогою колекції ***WshNamed***;
* виділити тільки ті параметри, які не мають імен (безіменні параметри) за допомогою колекції ***WshUnnamed***.

Далі приведений приклад сценарію, в якому на екран виводяться загальна кількість параметрів командного рядка, кількість іменних і безіменних аргументів, а також значення кожної з цих груп параметрів.

***var i, Arg, objArgs, s, objNamedArgs, objUnnamedArgs; //Оголошуємо змінні***

***var objArgs = WScript.Arguments; //Створюємо об'єкт WshArguments***

***//Визначаємо загальну кількість аргументів***

***s = "Total arguments:" + objArgs.Count() + "\n";***

***for (var i=0; i<objArgs.length; i++) {***

***s += objArgs(i) + "\n";***

***}***

***objUnnamedArgs = objArgs.Unnamed //Створюємо об'єкт WshUnnamed***

***//Визначаємо кількість безіменних аргументів***

***s += "\nUnnamed arguments:" + objUnnamedArgs.length + "\n";***

***for (var i=0; i<objUnnamedArgs.length; i++) {***

***s += objUnnamedArgs(i) + "\n";***

***}***

***objNamedArgs = objArgs.Named; //Створюємо об'єкт WshNamed***

***//Визначаємо кількість іменних аргументів***

***s += "\nNamed arguments:" + objNamedArgs.length + "\n";***

***//Перевіряємо, чи існує аргумент /Ім'я:***

***if (objNamedArgs.Exists("Name"))***

***s += objNamedArgs("Name") + "\n";***

***//Перевіряємо, чи існує аргумент /Comp:***

***if (objNamedArgs.Exists("Comp"))***

***s +=objNamedArgs ("Comp") + "\n";***

***WScript.Echo(s); //Виводимо сформовані рядки***

## Об'єкт WshEnvironment

Об'єкт ***WshEnvironment*** дозволяє отримати доступ до колекції, що містить змінні середовища заданого типу (змінні середовища операційної системи, змінні середовища користувача або змінні середовища поточного командного вікна). Цей об'єкт можна створити за допомогою властивості Environment об'єкта ***WshShell*** або однойменного його методу:

***WshShell = WScript.CreateObject("WScript.Shell");***

***WshSysEnv = WshShell.Environment;***

***WshUserEnv = WshShell.Environment("User");***

Об'єкт ***WshEnvironment*** має властивість ***Length***, в якому зберігається число елементів в колекції (кількість змінних середовища), і методи ***Count*** і ***Item***. Для того щоб отримати значення певної змінної середовища, в якості аргументу методу ***Item*** вказується ім'я цієї змінної в подвійних лапках. У наступному прикладі ми виводимо на екран значення змінної середовища ***PATH***.

***var WshShell, WshSysEnv;***

***WshShell = WScript.CreateObject("WScript.Shell");***

***WshSysEnv = WshShell.Environment;***

***WScript.Echo("System Path: ",WshSysEnv.Item("PATH"));***

Можна також просто вказати ім'я змінної в круглих дужках після імені об'єкта:

***WScript.Echo("Системний шлях: ", WshSysEnv("PATH"));***

Крім цього в об'єкта ***WshEnvironment*** є метод ***Remove(strName)***, який видаляє задану змінну середовища.

## Об'єкт WshSpecialFolders

При установці Windows завжди автоматично створюються кілька спеціальних папок (наприклад, папка для робочого столу (Desktop) або папка для меню Пуск (Start)), шлях до яких згодом може бути тим чи іншим способом змінений. Об'єкт WshSpecialFolders забезпечує доступ до колекції, що містить шляху до спеціальних папках Windows; завдання шляхів до таких папок може знадобитися, наприклад, для створення безпосередньо зі сценарію ярликів на робочому столі.

Об'єкт ***WshSpecialFolders*** створюється за допомогою властивості ***SpecialFolders*** об'єкта ***WshShell***:

***WshShell = WScript.CreateObject("WScript.Shell");***

***WshSpecFold = WshShell.SpecialFolders;***

Далі наведено сценарій, який формує список всіх наявних в системі спеціальних папок.

***var WshShell, WshFldrs, SpecFldr, s; //Оголошуємо змінні***

***//Створюємо об'єкт WshShell***

***WshShell = WScript.CreateObject("Wscript.Shell");***

***//Створюємо об'єкт WshSpecialFolders***

***WshFldrs = WshShell.SpecialFolders;***

***s = "List of all special folders: " + "\n\n";***

***//Перебираємо всі елементи колекції WshFldrs***

***for (var i=0; i<WshFldrs.length; i++) {***

***s+=WshFldrs(i)+"\n"; //Формуємо рядок зі значеннями аргументів***

***}***

***WScript.Echo(s);***

## Лекція №7. «Введення в оболонку командного рядка Windows PowerShell. Типи команд PowerShell. Робота з файловою системою. Конвеєризація команд у PowerShell. Вибірка, сортування та групування об’єктів.»

## Введення в оболонку командного рядка Windows PowerShell

Нова оболонка ***Windows PowerShell*** (попередньо вона була названа ***Monad***) була задумана розробниками ***Microsoft*** як більш потужне середовище для написання сценаріїв та роботи з командного рядка. Розробники ***PowerShell*** переслідували кілька цілей. Головна і сама амбітна з них - створити середовище складання сценаріїв, яка найкращим чином підходила б для сучасних версій операційної системи Windows і була б більш функціональною, що розширюється і простий у використанні, ніж будь-який аналогічний продукт для будь-якої іншої операційної системи. В першу чергу це середовище повинна була підходити для вирішення завдань, що стоять перед системними адміністраторами (тим самим Windows отримала б додаткову перевагу в боротьбі за сектор корпоративних платформ), а також задовольняти вимогам розробників програмного забезпечення, надаючи їм засоби для швидкої реалізації інтерфейсів управління до створюваним додатків.

Для досягнення цих цілей були вирішені наступні завдання:

Забезпечення прямого доступу з командного рядка до об'єктів ***COM***, ***WMI*** і ***.NET***. У новій оболонці присутні команди, що дозволяють в інтерактивному режимі працювати з COM-об'єктами, а також з екземплярами класів, визначених в інформаційних схемах ***WMI*** і ***.NET***.

Організація роботи з довільними джерелами даних в командному рядку за принципом файлової системи. Наприклад, навігація по системного реєстру або сховища цифрових сертифікатів виконується з командного рядка за допомогою аналога команди CD інтерпретатора ***Cmd.exe***.

Розробка інтуїтивно зрозумілої уніфікованої структури вбудованих команд, заснованої на їх функціональне призначення. У новій оболонці імена всіх внутрішніх команд (в ***PowerShell*** вони називаються Командлети) відповідають шаблону "дієслово-іменник", наприклад, ***Get-Process*** (отримати інформацію про процес), ***Stop-Service*** (зупинити службу), ***Clear-Host*** (очистити екран консолі) і т.д. Для однакових параметрів внутрішніх команд використовуються стандартні імена, структура параметрів у всіх командах ідентична, всі команди обробляються одним синтаксичним аналізатором. В результаті полегшується запам'ятовування і вивчення команд.

Забезпечення можливості розширення вбудованого набору команд. Внутрішні команди ***PowerShell*** можуть доповнюватися командами, які створюються користувачем. При цьому вони повністю інтегруються в оболонку, інформація про них може бути отримана з стандартної довідкової системи ***PowerShell***.

Організація підтримки знайомих команд з інших оболонок. У ***PowerShell*** на рівні псевдонімів власних внутрішніх команд підтримуються найбільш часто використовувані стандартні команди з оболонки ***Cmd.exe*** і Unix-оболонок. Наприклад, якщо користувач, який звик працювати з Unix-оболонкою, виконає ***ls***, то він отримає очікуваний результат: список файлів в поточному каталозі (те ж саме відноситься до команди ***dir***).

Розробка повноцінної вбудованої довідкової системи для внутрішніх команд. Для більшості внутрішніх команд в довідковій системі дано докладний опис і приклади використання. У будь-якому випадку вбудована довідка по будь-якій внутрішній команді буде містити короткий опис всіх її параметрів.

Реалізація автоматичного завершення при введенні з клавіатури імен команд, їх параметрів, а також імен файлів і папок. Дана можливість значно спрощує і прискорює введення команд з клавіатури.

Головною особливістю середовища ***PowerShell***, що відрізняє її від всіх інших оболонок командного рядка, є те, що одиницею обробки і передачі інформації тут є об'єкт, а не рядок тексту.

### Відмінність PowerShell від інших оболонок - орієнтація на об'єкти

При розробці будь-якої мови програмування одним з основних є питання про те, які типи даних і яким чином будуть в ньому представлені. При створенні ***PowerShell*** розробники вирішили не винаходити нічого нового і скористатися уніфікованої об'єктної моделлю ***.NET***. Даний вибір був зроблений з кількох причин.

По-перше, платформа ***.NET*** повсюдно використовується при розробці програмного забезпечення для Windows і представляє, зокрема, загальну інформаційну схему, за допомогою якої різні компоненти операційної системи можуть обмінюватися даними один з одним.

По-друге, об'єктна модель ***.NET*** є такою, що самодокумнтується: кожен об'єкт ***.NET*** містить інформацію про свою структуру. При інтерактивній роботі це дуже корисно, тому що з'являється можливість безпосередньо з командного рядка виконати запит до певного об'єкту і побачити опис його властивостей і методів, тобто зрозуміти, які саме маніпуляції можна виконати з цим об'єктом, не вивчаючи додаткової документації з його описом.

По-третє, працюючи в оболонці з об'єктами, можна за допомогою їх властивостей і методів легко отримувати потрібні дані, не займаючись розбором і аналізом символьної інформації, як це відбувається у всіх традиційних тексто-орієнтованих оболонках командного рядка. Розглянемо приклад. У Windows XP є консольна утиліта ***tasklist.exe***, яка видає інформацію про процеси запущених в системі:

***C:\> tasklist***

Припустимо, що ми в командному файлі інтерпретатора ***Cmd.exe*** за допомогою цієї утиліти хочемо визначити, скільки оперативної пам'яті витрачає процес ***kavsvc.exe***. Для цього потрібно виділити з вихідного потоку команди ***tasklist*** відповідний рядок, витягти з нього підрядок, що містить необхідну кількість і прибрати проміжки між розрядами (при цьому слід врахувати, що в залежності від налаштувань операційної системи роздільником розрядів може бути не проміжок, а інший символ). У ***PowerShell*** аналогічна задача вирішується за допомогою команди ***get-process***, яка повертає колекцію об'єктів, кожен з яких відповідає одному запущеному процесу. Для визначення пам'яті, що витрачається процесом ***kavsvc.exe***, немає необхідності в додаткових маніпуляціях з текстом, досить просто взяти значення властивості WS об'єкта, яке відповідає даному процесу.

Нарешті, об'єктна модель ***.NET*** дозволяє ***PowerShell*** безпосередньо використовувати функціональність різних бібліотек, які є частиною платформи ***.NET***. Наприклад, щоб дізнатися, яким днем ​​тижня було 9 листопада 1974 року народження, в ***PowerShell*** можна виконати наступну команду:

***(Get-date "09.11.1974").DayOfWeek***

В цьому випадку команда ***get-date*** повертає .NET-об'єкт ***DateTime***, що має властивість, при зверненні до якого обчислюється день тижня для відповідної дати. Таким чином, розробникам ***PowerShell*** не потрібно створювати спеціальну бібліотеку для роботи з датами і часом - вони просто беруть готове рішення в ***.NET***.

### Запуск оболонки. Виконання команд

Встановивши оболонку в системі, можна почати новий інтерактивний сеанс. Для цього слід натиснути на кнопку ***Пуск (Start)***, відкрити меню ***Всі програми (All Programs)*** і вибрати елемент ***Windows PowerShell***. Інший варіант запуску оболонки - пункт ***Виконати ... (Run)*** в меню ***Пуск (Start)***, ввести ім'я файлу ***powershell*** і натиснути кнопку OK.

В результаті відкриється нове командне вікно із запрошенням вводити команди.

Виконаємо першу команду в ***PowerShell***. Нехай це буде щось знайоме, наприклад ***dir*** (команди в ***PowerShell*** обробляються без урахування регістру). На екран буде виведено список файлів в поточному каталозі:

***PS C:\Documents and Settings\User> dir***

***Каталог: Microsoft.PowerShell.Core\FileSystem:: C:\ Documents and Settings\User***

***Mode LastWriteTime Length Name***

***---- ------------- ------ ----***

***d---s 14.12.2007 10:10 Cookies***

***d---- 12.05.2005 17:16 DoctorWeb***

***d---- 06.11.2004 13:03 Phone Browser***

***d---s 22.09.2004 23:49 UserData***

***d-r-- 04.10.2004 15:33 Головне меню***

***d-r-- 05.12.2007 0:49 Вибране***

***d-r-- 31.10.2007 21:03 Мої документи***

***d---- 05.12.2007 10:54 Робочий стіл***

***-a--- 24.05.2006 21:22 8304 gsview32.ini***

## 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 є 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.

Даний тип команд компілюється в динамічну бібліотеку (DLL) і подгружается до процесу ***PowerShell*** під час запуску оболонки (тобто самі по собі командлети не можуть бути запущені як додатки, але в них містяться виконувані об'єкти). Так як компілює код подгружается до процесу оболонки, даний тип команд виконується найбільш ефективно. Командлети - це аналог внутрішніх команд традиційних 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 проводиться один раз при її оголошенні.

Сценарій - це блок коду на мові ***PowerShell***, що зберігається в зовнішньому файлі з розширенням ***ps1***. Аналіз синтаксису сценарію проводиться при кожному його запуску.

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

### Імена та структура командлетів

Як уже було відзначено вище, в ***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 при використанні їх композиції (конвеєризації об'єктів між командлетами).

Імена командлетів завжди відповідають шаблону "дієслово-іменник", де дієслово задає певну дію, а іменник визначає об'єкт, над яким цю дію буде здійснено.

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

***PS C:\> Get-Command***

***CommandType Name Definition***

***----------- ---- ----------***

***Cmdlet Add-Content Add-Content [-Path] <String []> [-Value] <Object [...***

***Cmdlet Add-History Add-History [[-InputObject] <PSObject []>] [-Pass ...***

***Cmdlet Add-Member Add-Member [-MemberType] <PSMemberTypes> [-Name] ...***

***Cmdlet Add-PSSnapin Add-PSSnapin [-Name] <String []> [-PassThru] [-Ve ...***

***Cmdlet Clear-Content Clear-Content [-Path] <String []> [-Filter <Strin ...***

***Cmdlet Clear-Item Clear-Item [-Path] <String []> [-Force] [-Filter ...***

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

Командлети можуть мати параметри, до яких можна звернутися по імені, перед яким ставиться дефіс (-), або по позиції (в останньому випадку інтерпретація параметра буде виконуватися в залежності від його місця розташування в командному рядку).

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***.

### Автоматичне завершення команд

Перебуваючи в оболонці ***PowerShell***, можна ввести частину будь-якої команди, натиснути клавішу ***<Tab>*** і система спробує сама завершити цю команду.

Подібне автоматичне завершення спрацьовує, по-перше, для імен файлів і шляхів файлової системи. При натисканні клавіші ***<Tab> PowerShell*** автоматично розширить частково введений шлях файлової системи до першого знайденого збіги. При повторенні натискання клавіші ***<Tab>*** проводиться циклічний перехід за наявними можливостями вибору. Також в ***PowerShell*** реалізована можливість автоматичного завершення шляхів файлової системи на основі шаблонних символів (***?*** І ***\****). Наприклад, якщо ввести команду ***cd c:\pro\* files*** і натиснути клавішу ***<Tab>***, то в рядку введення з'явиться команда ***cd 'C:\Program Files'***.

По-друге, в ***PowerShell*** реалізовано автозавершення імен командлетів і їх параметрів. Якщо ввести першу частину імені командлет (дієслово) і дефіс, натиснути після цього клавішу ***<Tab>***, то система підставить ім'я першого підходящого командлет (наступний підходящий варіант імені вибирається шляхом повторного натискання ***<Tab>***). Аналогічним чином автозавершення спрацьовує для частково введених імен параметрів командлет: натискаючи клавішу ***<Tab>***, ми будемо циклічно перебирати підходящі імена.

Нарешті, ***PowerShell*** дозволяє автоматично завершувати імена використовуваних змінних (об'єктів) та імена властивостей об'єктів.

### Псевдоніми команд

Механізм псевдонімів, реалізований в оболонці ***PowerShell***, дає можливість користувачам виконувати команди по їх альтернативним іменам (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 тип 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*** скорочується до ***s***, іменник ***Location*** скорочується до ***l*** і т.д. Таким чином, для командлету ***Set-Location*** відповідає псевдонім ***sl***, а командлету ***Get-Location*** - псевдонім ***gl***.

Переглянути список всіх пседонімов, 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 по одному командлету можна отримати за допомогою параметра ***?*** (Знак питання), зазначеного після імені цього командлета. Наприклад:

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

Більш детальну інформацію можна отримати за допомогою спеціального командлету Get-***Help*** з параметрами ***Detailed*** або ***Full*** (наприклад, ***Get-Help Get-Process -Full***). Запустивши цей командлет з параметром ***\****, можна побачити всі доступні розділи довідкової системи.

Командлет ***Get-Help*** дозволяє переглядати довідкову інформацію не тільки про різні Командлети, а й про синтаксис мови ***PowerShell***, про псевдонімах, про різні аспекти роботи оболонки функцій і т.д. Список тем, обговорення яких представлено з довідковій службі ***PowerShell***, можна побачити в такий спосіб:

***PS C:\> Get-Help about\_\****

***Name Category Synopsis***

***---- -------- --------***

***about\_alias HelpFile Використання альтернативних імен ко ...***

***about\_arithmetic\_operators HelpFile Оператори, які використовуються в ко ...***

***about\_array HelpFile Компактна структура розміщення елем ...***

***about\_assignment\_operators HelpFile Оператори, які використовуються в оболонці W ...***

***about\_associative\_array HelpFile Компактна структура даних для храни ...***

***about\_automatic\_variables HelpFile Змінні, що автоматично задаються ...***

***about\_break HelpFile Інструкція для негайного завершено ...***

***about\_command\_search HelpFile Як оболонка Windows PowerShell обна ...***

***about\_command\_syntax HelpFile Формат команд в Windows PowerShell***

Таким чином, щоб прочитати докладну інформацію щодо використання масивів в ***PowerShell***, потрібно виконати наступну команду: ***Get-Help about\_array***.

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

**Робота з файловою системою у PowerShell**

***Навігація у файловій системі***

В оболонці **cmd.exe** зміна поточного каталогу проводиться за допомогою команди ***cd***. У **PowerShell** команда ***cd*** має таке ж значення, при цьому вона є стандартним псевдонімом командлета ***Set-Location***. Інши псеводніми – ***chdir***, ***sl***. 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). При цьому потрібно враховувати наступний нюанс. Людина, яка часто користувалась командою ***cd*** в оболонці ***cmd.exe***, швидше за все, буде машинально набирати команди типу ***cd..*** або ***cd\*** без додаткових проміжків. У **PowerShell** це викличе помилку.

Ця помилка пов'язана з тим. що в **PowerShell** параметри команди завжди повинні відділятися від імені самої команди проміжком. 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***. Інши псеводніми – ***ls***, ***gci***.

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 С:\> dir С:\Windows\\*.log***

Параметр ***-Exclude*** дозволяє задати маску файлів, які не будуть оброблюватись 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, при якому командлет ***dir*** відображає не тільки вміст зазначеного каталогу, але і всіх його підкаталогів:

***PS С:\> dir 'Documents and Settings' –Recurse***

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

***PS С:\> dir C:\Windows –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 ***С:*** каталог з ім'ям ***test folder***:

***PS С:\> New-Item -Path C:\test\_folder -Type "directory"***

При створенні файлу в нього відразу можна записати рядок, вказавши його як значення параметра ***-Value***, наприклад:

***PS С:\> New-Item -Path C:\test\_file.txt -Type "file" -Value "Test"***

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

***PS С:\> New-Item -Path C:\test\_file.txt -Type "file" -Value "Test"***

то виникне помилка.

Для перезапису існуючого файлу при створенні потрібно вказати параметр ***-Force***:

***PS С:\> 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 (при цьому рядки відображаються на екрані). Наприклад:

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

Параметр ***-Encoding*** командлет ***Get-Content*** дозволяє явно вказувати кодування файлу для коректної обробки його вмісту. Допустимі значення даного параметра: ***Unicode***, ***Byte***, ***BigEndianUnicode***, ***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 С:\> Get-Content C:\Windows\win.ini -TotalCount 5***

**Запис файлів**

Записати дані у зовнішні файли можна за допомогою операторів перенаправлення (***>*** і ***>>***) і командлет ***Out-File***. При цьому командлет ***Out-File*** намагається форматувати в файл об'єкти, що записуються . Якщо потрібно просто записати в файл текстову інформацію (без додаткового форматування), то краще скористатися командлетом ***Set-Content***.

Дані 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***, які мають псевдонім ***copy***. Шлях до файлів, що копіюються, при цьому вказується в якості значення параметра ***-Path*** (даний параметр використовується за замовчуванням), а шлях до цільового каталогу, в який потрібно скопірювати файли, задається значенням параметра ***-Destination***. Наприклад, наступна команда скопіює файл ***styles.css*** з кореневого каталогу диска з: в каталог ***C:\test\_folder***:

***PS С:\> 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, в тому числі від ***cmd.exe***). Наприклад:

***PS С:\> copy C:\script -Destination C:\test\_folder -PassThru***

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

***PS С:\> copy C:\script -Destination C:\test\_folder -Recurse -PassThru***

Можна копіювати не всі файли з каталогу, а тільки відповідні певній масці. При цьому маску 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*** по конвеєру. Наприклад, наступна команда копіює всі файли з розширенням ***psl*** з каталогу ***C:\script*** і всіх його підкаталогів в папку ***C:\test\_folder***:

***PS С:\> 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 С:\> 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 С:\> 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 цих параметрів можна опускати (в цьому випадку першим має зазначатися значення параметра ***-Path***). Наприклад, створимо файл ***c:\l.tmp*** і перейменуємо його у файл ***2.tmp***:

***PS С:\> 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 С:\> 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 слід скористатися командлетом ***Move-item*** (псевдонім ***move***). Значення параметра ***-Path*** даного командлету задає шлях до файлів або каталогів для переміщення (в цьому шляху допускається використання символів узагальнення), а значення параметра ***–Destination*** - шлях до каталогу. куди будуть переміщені ці файли або каталоги. Результат перемещення можна побачити на екрані, вказавши параметр ***-PassThru***. Наприклад, наступна команда перенесе у кореневий каталог диску ***С:*** каталог ***C:\test\_folder\folderl*** з усім його вмістом:

***PS C:\> Move-Item -Path C:\test\_folder\folderl C:\ -PassThru***

Видаляти об'єкти файлової системи можна за допомогою командлета ***Remove-Item*** (псевдонім ***del***). Значення параметра ***-Path*** цього командлета 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 ***С:\test\_folder***:

***PS С:\> del С:\test\_folder\\*.psl***

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

***PS С:\> del С:\test\_folder\\****

**Всі результати виконання команд у PowerShell є об’єктами .NET**

Раніше ми вже неодноразово казали про те, що всі дії в оболонці **PowerShell** пов'язані з операціями над об'єктами.

Незайвим буде нагадати, що об'єкт-це сукупність даних (властивості об'єкта) і способів роботи з цими даними (методи об'єкта). Конкретна структура об'єкта (складу властивостей і методів) задається його типом (наприклад, будь-якому файлу на жорсткому диску відповідає об'єкт типу ***Fileinfo***). Набір типів, що використовуються в **PowerShell**, базується на типах уніфікованої платформи .NET Framework, повсюдно використовується в сучасних версіях операційної системи Windows.

Властивість об'єкта - це відомості про стан або параметри. Наприклад. у об'єкта ***Fileinfo*** є властивість ***Length*** (довжина), що відповідає розміру файлу, який представлений даним об'єктом.

Метод об'єкта є дією, яку можна здійснювати над елементом, представленим даним об'єктом. Наприклад, у об'єкта ***Fileinfo*** є метод ***соруТо***, за допомогою якого можна скопіювати файл (при виклику цього методу відбувається копіювання представленого об'єктом файлу на рівні файлової системи).

Властивості і методи об'єктів використовуються в командлетах **PowerShell** для виконання різних дій і роботи з даними. При цьому в **PowerShell** підтримується запозичений з інших інтерфейсів командного рядка механізм конвеєризації або композиції команд, що значно підвищує ефективність роботи.

**Конвеєризація об'єктів в PowerShell**

У більшості оболонок командного рядка, включаючи **cmd.exe**, під конвеєризацією розуміється об'єднання (композиція) кількох команд шляхом послідовного перенаправлення вихідного потоку однієї команди  у вхідний потік іншої, що дозволяє передавати текстову інформацію між різними процесами.

Механізм композиції команд є, ймовірно, найбільш цінну концепцію, яка використовується в інтерфейсах командного рядка. Конвеєри не тільки знижують зусилля, прикладені при введенні складних команд, але і полегшують відстеження виконуваних командами дій. Корисною рисою конвеєрів є те. що вони не залежать від числа переданих елементів, так як конвеєр діє на кожен елемент окремо. Крім того, кожна команда в конвеєрі (звана елементом конвеєра) зазвичай передає свій висновок наступній команді в конвеєрі, елемент за елементом. Завдяки цьому, як правило, знижується споживання ресурсів для складних команд і з'являється можливість отримувати інформацію, що виводиться негайно.

В оболонці **PowerShell** також дуже широко використовується механізм конвеєризації команд, проте тут по конвеєру передається не потік тексту, як у всіх інших оболонках, а **об'єкти**. При цьому з елементами конвеєра можна виробляти різні маніпуляції: фільтрувати об'єкти але певним критерієм, сортувати і групувати об'єкти, змінювати їх структуру.

**Конвеєр в PowerShell** - це послідовність команд, розділених між собою знаком ***|*** (Вертикальна риса). 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 при необхідності витягуються певні елементи.

Передача даних 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, так як зазвичай виведення команди більше орієнтоване на візуальне сприйняття людиною (це природно для інтерактивного режиму роботи), а не на зручність подальшого синтаксичного розбору.

При передачі по конвеєру об'єктів цієї проблеми не виникає, тут потрібна інформація витягується з елемента конвеєра простим зверненням до відповідної властивості об'єкта. Однак тепер виникає нове запитання: як можна дізнатися, які саме властивості є у об'єктів, що передаються по конвеєру? Адже при виконанні того чи іншого командлету ми на екрані бачимо тільки одну або кілька колонок відформатованого тексту. Наприклад, запустимо командлет ***Get-Process***, який виводить інформацію про запущені в системі процеси:

***PS С:\> Get-Process***

***Handles NPM(K) PM(K) WS(K) VM(M) CPU(s) Id ProcessName***

***------- ------ ----- ----- ----- ------ -- -----------***

***386 41 116220 130400 269 20,42 5808 AcroRd32***

***234 17 9368 15952 105 0,34 5952 AcroRd32***

***287 18 5572 12176 71 1364 AGSService***

***--------------------------------------------------------------------***

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 саме властивість для цього потрібно перевіряти у виведених об'єктів?

Для відповіді на ці запитання потрібно, перш за все, навчитися досліджувати структуру об'єктів **PowerShell**, дізнаватися, які властивості і методи є у цих об'єктів.

**Командлет Get-Member – перегляд структури об’єктів**

Для аналізу структури об'єкта, що повертається певною командою, простіше, всього направити цей об'єкт по конвеєру на командлет ***Get-Member*** (псевдонім ***gm***), наприклад:

***PS С:\> 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***

***Disposed Event System.EventHandler Disposed(System.Object, System.EventArgs)***

***---------------------------------------------------------------------------***

В результаті на екрані ми бачимо, який .NET-тип мають об'єкти, які повертаються в ході роботи досліджуваного командлет (в нашому прикладі це тип ***System.Diagnostic.Process***), а також повний список елементів об'єкта. При цьому на екран виводиться дуже багато елементів різних типів (імена і псевдонім властивостей, імена методів і т. д.), і такий довгий список стає незручно переглядати. Командлет ***Get-Member*** має параметр ***-MemberТуре***, що дозволяє перерахувати тільки елементи об'єкта певного типу. Наприклад, для виведення тільки елементів об'єкта, що є властивостями цього об'єкта, використовується параметр ***-MemberType*** зі значенням ***Property***:

***PS С:\> Get-Process | Get-Member -MemberType Property***

***TypeName: System.Diagnostics.Process***

***Name MemberType Definition***

***---- ---------- ----------***

***BasePriority Property System.Int32 BasePriority {get;}***

***Container Property System.ComponentModel.IContainer Container {get;}***

***EnableRaisingEvents Property System.Boolean EnableRaisingEvents {get;set;}***

***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;}***

***MachineName Property System.String MachineName {get;}***

***MainModule Property System.Diagnostics.ProcessModule MainModule {get;}***

***MainWindowHandle Property System.IntPtr MainWindowHandle {get;}***

***MainWindowTitle Property System.String MainWindowTitle {get;}***

***---------------------------------------------------------------------------***

Як бачите, процесам операційної системи відповідають об'єкти, що мають дуже багато властивостей, на екран же при роботі командлету ***Get-Process*** виводяться лише кілька з них. Насправді способи відображення в оболонці **PowerShell** об'єктів різних типів задаються декількома файлами з розширенням **pslxml** в форматі ХМL, які знаходяться в каталозі, де встановлений файл **powershell.exe** (шлях до цього каталогу зберігається в змінної ***$PSHome***). Список цих файлів можна отримати за допомогою наступної команди:

***PS С:\> dir $pshome\\*format\*.pslxml***

Зокрема, правило форматування об'єкта типу ***System.Diagnostic.Process*** знаходиться в файлі **dotnettypes.format.pslxml**. Редагувати безпосередньо конфігураційні файли не рекомендується, а в разі необхідності можна створити власні файли форматування і за допомогою командлета ***Update-FormatData*** включити їх до складу файлів, що автоматично завантажуються.

Тепер, коли ми знаємо, які властивості мають об'єкти, що передаються по конвеєру, перейдемо до розгляду можливих операцій над елементами конвеєра.

**Командлет Where-Object – фільтрація об'єктів**

У **PowerShell** підтримується можливість фільтрації об'єктів в конвеєрі, тобто видалення з конвеєра об'єктів, які задовольняють певній умові. Дану функціональність забезпечує командлет ***Where-Object***, що дозволяє перевірити кожний об'єкт, що проходить через конвеєр, і передати його далі по конвеєру лише в тому випадку, якщо об'єкт задовольняє умовам перевірки.

Умова перевірки в ***Where-Objeсt*** задається в вигляді блоку сценарію (**scriptblock**) - однієї або декількох команд **PowerShell,** укладених у фігурні дужки ***{}***. Блок сценарію вказується після імені командлету ***Where-Object***. Результатом виконання блоку сценарію в командлеті ***Where-Object*** має бути значення логічного типу: ***$Тruе*** (істина, в цьому випадку об'єкт проходить далі по конвеєру) або ***$False*** (брехня, в цьому випадку об'єкт далі по конвеєру не передається).

Наприклад, для виведення інформації про зупинені служби в системі (об'єкти, які повертаються командлетом ***Get-Service***, у яких властивість ***Status*** дорівнює "***stopped***") можна використовувати наступний конвеєр:

***PS C:\> Get-Service | Where-Object {$\_.Status -eq "Stopped"}***

***Status Name DisplayName***

***------ ---- -----------***

***Stopped AeLookupSvc Application Experience***

***Stopped ALG Application Layer Gateway Service***

***Stopped Apache2.2-Zend Apache2.2-Zend***

***Stopped AppIDSvc Application Identity***

***Stopped AppMgmt Application Management***

***Stopped aspnet\_state ASP.NET State Service***

***Stopped AudioEndpointBu... Windows Audio Endpoint Builder***

***Stopped AudioSrv Windows Audio***

***-----------------------------------------------------------***

Інший приклад - залишимо в конвеєрі тільки ті процеси, у яких значс ня ідентифікатора (властивість ***id***) більше 4000:

***PS С:\> Get-Process | Where-Object {$\_.Id -gt 4000}***

***Handles NPM(K) PM(K) WS(K) VM(M) CPU(s) Id ProcessName***

***------- ------ ----- ----- ----- ------ -- -----------***

***434 44 116748 131760 277 25,09 5808 AcroRd32***

***315 22 10160 17812 112 0,55 5952 AcroRd32***

***243 28 7928 5588 124 4344 AvastUI***

***227 26 7628 5776 119 54,89 4984 AvastUI***

***213 11 6220 8864 924 5172 cache***

***181 16 13312 73336 205 2,53 5944 chrome***

***324 39 97320 242216 956 22,59 6232 chrome***

***199 23 32884 89592 770 3,84 6332 chrome***

***76 9 2952 6200 76 0,02 6544 chrome***

***---------------------------------------------------------------------***

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

Як можна зрозуміти з прикладів, в блоці сценарію використовуються спеціальні оператори порівняння. Основні оператори порівняння наведені в табл. 7.1.

У **PowerShell** для операторів порівняння не використовуються звичайні символи “***>***” або “***<***”, так як в командному рядку вони зазвичай означають перенаправлення введення/виведення.

Табл. 7.1. 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*** |
| ***-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 символи узагальнення у другому операнді | ***"file.doc" -like "f \*.doc"*** |
| ***-notlike*** | порівняння на розбіжність з урахуванням символів узагальнення в другому операнді | ***"fiie.doc" -notlike "f\*.rtf"*** |
| ***-contains*** | містить | ***1,2,3 -contains 1*** |
| ***-notcontains*** | не містить | ***1,2,3 -notcontains 4*** |

Оператори порівняння можна з'єднувати один з одним за допомогою логічних операторів (табл. 8.2).

Табл. 8.2. 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)*** |

**Командлет Sort-Object – сортування об'єктів**

Сортування елементів конвеєра - еше одна операція, яка часто застосовується при конвеєрній обробці об'єктів. Дану операцію здійснює командлет ***Sort-Object***: йому передаються імена властивостей, за якими потрібно провести сортування об'єктів, що проходять по конвеєру, а він повертає дані, впорядковані за значеннями цих властивостей.

Наприклад, для виведення списку запущених в системі процесів, впорядкованого за іх ідентифікаторами (властивість ***Id***). можна скористатись наступним конвеєром:

***PS С:\> Get-Process | Sort-Object -Property Id***

***Handles NPM(K) PM(K) WS(K) VM(M) CPU(s) Id ProcessName***

***------- ------ ----- ----- ----- ------ -- -----------***

***0 0 0 24 0 0 Idle***

***772 0 112 320 3 4 System***

***796 61 34868 23932 455 12 AvastSvc***

***38 2 572 1192 4 304 smss***

***1042 16 2796 5320 50 388 csrss***

***85 10 1644 4596 48 440 wininit***

***71 9 1860 3964 41 448 csrss***

***80 7 2020 14912 49 452 svchost***

***75 6 1852 4632 25 488 winlogon***

***286 13 5440 10124 42 536 services***

***-------------------------------------------------------------------***

Параметр ***-Property*** в командлеті ***Sort-Object*** використовується за замовчуванням, тому ім'я цього параметра можна не вказувати. Для сортування в зворотному порядку використовується параметр ***-Descending***:

***PS С:\> Get-Process | Sort-Object Id –Descending***

***Handles NPM(K) PM(K) WS(K) VM(M) CPU(s) Id ProcessName***

***------- ------ ----- ----- ----- ------ -- -----------***

***80 9 2952 6200 76 0,09 6544 chrome***

***40 6 2248 5552 61 1,11 6352 conhost***

***199 23 37252 92984 772 4,69 6332 chrome***

***324 39 109988 253168 962 43,31 6232 chrome***

***55 2 2512 4884 10 6040 VirtualBox***

***377 25 13068 20796 119 6032 spoolsv***

***313 22 10120 17808 111 0,56 5952 AcroRd32***

***201 17 17572 77668 208 2,61 5944 chrome***

***50 2 1520 2952 9 5940 VirtualBox***

***---------------------------------------------------------------------***

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

***PS С:\> Get-Process | Where-Object {$\_.Id -gt 1000} | Sort-Object Id –Descending***

***Handles NPM(K) PM(K) WS(K) VM(M) CPU(s) Id ProcessName***

***------- ------ ----- ----- ----- ------ -- -----------***

***80 9 2924 6180 74 0,09 6544 chrome***

***40 6 2268 5580 61 1,31 6352 conhost***

***199 23 37512 93292 772 4,78 6332 chrome***

***324 40 112556 254648 964 45,09 6232 chrome***

***55 2 2512 4884 10 6040 VirtualBox***

***377 26 13120 20812 119 6032 spoolsv***

***313 22 10120 17808 111 0,56 5952 AcroRd32***

***201 17 17572 77668 208 2,61 5944 chrome***

***50 2 1520 2952 9 5940 VirtualBox***

***461 44 122588 137688 299 45,80 5808 AcroRd32***

***204 18 9400 17460 130 0,77 5528 WinRAR***

***1070 66 114232 125912 366 1 564,09 5220 googledrivesync***

***213 11 6220 8864 924 5172 cache***

***157 8 5432 13840 45 5072 OSPPSVC***

***890 62 69020 69260 309 63,42 5068 explorer***

***636 23 3236 23020 255 5052 csrss***

***117 7 2268 5660 31 5040 winlogon***

***465 29 12276 27624 239 1,20 5028 RdrCEF***

***974 18 14724 23708 116 5008 VBoxSVC***

**Командлет Select-Object – виділення об’єктів та властивостей**

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 по конвеєру.

Для виділення з конвеєра декількох перших або останніх об'єктів слід скористатися відповідно 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 С:\> 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***

Розберемо роботу даного конвеєра команд. Перший командлет в конвеєрі (***Get-Process***) повертає масив об'єктів, відповідних запущеним в системі процесам. Другий командлет ***Sort-Object*** впорядковує об'єкти, проходять по конвеєру, 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, в якому були б вказані тільки імена процесів і їх 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 С:\> 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 в якості параметрів командлет ***Select-Object***:

***PS С:\> Get-Process | Select-Object ProcessName, Id***

***ProcessName Id***

***----------- --***

***AcroRd32 5808***

***AcroRd32 5952***

***AGSService 1364***

***armsvc 1340***

***AvastSvc 12***

***AvastUI 4344***

***AvastUI 4984***

***--------------------------------------------------------------***

Подивимося тепер, який тип має об'єкт, що формується в конвеєрі комацдлетом ***Select-Object***, і які властивості є у цього об'єкта:

***PS С:\> Get-Process | Select-Object ProcessName, Id | Get-Member***

***TypeName: Selected.System.Diagnostics.Process***

***Name MemberType Definition***

***---- ---------- ----------***

***Equals Method bool Equals(System.Object obj)***

***GetHashCode Method int GetHashCode()***

***GetType Method type GetType()***

***ToString Method string ToString()***

***Id NoteProperty int Id=2404***

***ProcessName NoteProperty string ProcessName=armsvc***

Як бачите, вихідний об'єкт має тип ***System.Management.Automation.PSCustomObject*** (нагадаємо, що командлет ***Get-Process*** повертав об'єкти типу ***System.Diagnostics.Process***) І у нього є тільки дві властивості ***ProcessName*** і ***Id***. Це пов'язано з тим, що при використанні командлету ***Select-Object*** для вибору зазначених властивостей він копіює значення цих властивостей з об'єктів, що надходять по конвеєру йому на вхід, і створює нові об'єкти, які містять зазначені властивості зі скопійованими значеннями.

Командлет ***Select-Object*** може не тільки видаляти з об'єктів непотрібні властивості, але і додавати нові обчислювані властивості. Для цього нову властивість потрібно представити у вигляді хеш-таблиці, де перший елемент (ключ ***Name***) відповідає імені додається властивості, а другий елемент (ключ ***Expression***) - значенням цієї властивості для поточного елемента конвеєра.

Наприклад, результатом виконання наступного конвеєра команд стане масив об'єктів, що мають властивості ***ProcessName*** (ім'я запущеного процесу) і ***startMin*** (хвилина, коли був запущений процес):

***PS С:\> Get-Process | Select-Object ProcessName, @{Name="StartMin"; Expression={$\_.StartTime.Minute}}***

***ProcessName StartMin***

***----------- --------***

***armsvc***

***audiodg 47***

***AvastSvc***

***AvastUI 43***

***cache 55***

***chrome 43***

***cmd 55***

***conhost 11***

***-----------------------------***

Тут властивість ***startMin*** є обчислюваним, його значення для кожного елемента конвеєра задається блоком коду ***{$\_.StartTime.Minute}***, де змінна відповідає поточному об'єкту конвеєра.

**Командлет ForEach-Object – 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 по конвеєру командлету ***ForEach-Object***:

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

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

***PS D:\Gleb> $TotalLength***

***20336777***

**Командлет Group-Object – групування об’єктів**

Об'єкти, що проходять по конвеєру, можна згрупувати за значенням певних властивостей за допомогою командлета ***Group-Object***. В одну групу будуть потрапляти об'єкти, 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 С:\> 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)…***

***7 Microsoft Corporation {System.Diagnostics.Process (conhost)…***

***1 InterSystems {System.Diagnostics.Process (csystray)}***

***2 {System.Diagnostics.Process (googledrivesync)…***

***1 Alexander Roshal {System.Diagnostics.Process (WinRAR)}***

Як бачите, в колонці ***Count*** відображається кількість елементів у кожній з груп, а в колонці ***Group*** перераховані елементи, що входять до групи.

Якщо потрібно просто дізнатися кількість елементів в групах, можна запустити командлет ***Group-Object*** з параметром ***-NoElement***:

***PS С:\> Get-Process | Group-Object Company –NoElement***

***Count Name***

***----- ----***

***94***

***1 Avast Software s.r.o.***

***6 Google Inc.***

***7 Microsoft Corporation***

***1 InterSystems***

***2***

***1 Alexander Roshal***

**Командлет Measure-Object - вимірювання характеристик об'єктів**

У **PowerShell** є ще один корисний командлет ***Measure-Object***, призначений для виконання функцій агрегування (сума, вибір мінімального, максимального або середнього значення) над властивостями елементів в конвеєрі об'єктів.

Розглянемо приклад. Раніше ми вже знаходили спільний розмір файлів у певному каталозі, застосовуючи для цього командлет ***ForEach-Object***.

За допомогою командлета ***Measure-Object*** ми також зможемо знайти сумарний розмір файлів. Для цього потрібно вказати, що ***Measure-Object*** повинен для всіх елементів конвеєра підсумувати (параметр ***-Sum***) значення властивості ***Length***:

***PS C:\Documents and Settings\Gleb> dir | Measure-Object -Property Length –Sum***

***Count : 9***

***Average :***

***Sum : 20336777***

***Maximum :***

***Minimum :***

***Property : Length***

Результат буде виведений в поле ***Sum***. Для виконання інших операцій потрібно вказати відповідний параметр: ***-Average*** для знаходження середнього значення, ***-Minimum*** або ***-Maximum*** для знаходження мінімального або максимального значення відповідно:

***PS C:\Documents and Settings\Gleb> dir | Measure-Object -Property Length -Minimum -Maximum -Average -Sum***

***Count : 9***

***Average : 2259641,88888889***

***Sum : 20336777***

***Maximum : 13988219***

***Minimum : 33908***

***Property : Length***

Також за допомогою командлета ***Measure-Object*** можна отримувати статистичну інформацію про текстових файлах: кількість рядків, слів та символів.

**Лекція №8. «Виклик статичних методів. Управління виведенням команд у PowerShell. Робота зі змінними: прості змінні, масиви та хеш-таблиці»**

**Виклик статичних методів**

Іноді при роботі в **PowerShell** виникає необхідність скористатися методами, які визначені в класах (типах) **.NET**, не створюючи і не використовуючи екземпляри цих класів. Класи, що містять тільки такі методи, називаються статичними, так як вони не створюються, не знищуються і не змінюються. Зокрема, статичним є клас ***System.Math***, методи якого часто використовуються для математичних обчислень.

Для звернення до статичного класу його ім'я слід укласти в квадратні дужки, наприклад:

***PS С:\> [System.Math]***

***IsPublic IsSerial Name BaseType***

***-------- -------- ---- --------***

***True False Math System.Object***

Методи статичного класу також називаються статичними. Для перегляду доступних статичних методів класу потрібно передати ім'я цього класу (в квадратних дужках) по конвеєру командлету ***Get-Member*** з параметром ***-Static***:

***PS С:\> [System.Math] | Get-Member –Static***

***TypeName: System.Math***

***Name MemberType Definition***

***---- ---------- ----------***

***Abs Method static System.SByte Abs(System.SByte value), static System.Int16 Abs(System.Int16 value),...***

***Acos Method static double Acos(double d)***

***Asin Method static double Asin(double d)***

***Atan Method static double Atan(double d)***

***Atan2 Method static double Atan2(double y, double x)***

***BigMul Method static long BigMul(int a, int b)***

***Ceiling Method static decimal Ceiling(decimal d), static double Ceiling(double a)***

***Cos Method static double Cos(double d)***

***Cosh Method static double Cosh(double value)***

***---------------------------------------------------------------------------***

Для доступу до певного статичного методу або властивості використовуються дві йдуть двокрапки (***::***), а не крапка (***.***), як в звичайних об'єктах. Наприклад, для обчислення квадратного кореня з числа (статичного методу ***Sqrt***) і збереження результату в змінну використовується наступна конструкція:

***PS С:\> $а=[System.Math]::Sqrt(25)***

***PS С:\> $a***

***5***

**Управління виведенням команд в PowerShell**

В **PowerShell** є база даних (набір ХМL-файлів), що містить модулі форматування за замовчуванням для різних типів .NET-об'єктів. Ці модулі визначають, які властивості об'єкта відображуються при виведенні і в якому форматі: списку або таблиці. (Нагадаємо, що командлети **PowerShell** повертають .NET-об'єкти, які, як правило, не знають, яким чином відображати себе на екрані. Коли об'єкт досягає кінця конвеєра, **PowerShell** визначає його тип і шукає його в списку об'єктів, для яких визначено правило форматування. Якщо даний тип в списку виявлений, то до об'єкта застосовується відповідний модуль форматування; якщо немає, то **PowerShell** просто відображає властивості цього .NЕТ-об'єкта.

Також в **PowerShell** можна явно задавати правила форматування даних, що виводяться командлетами, і як і в командному інтерпретаторі **cmd.exe** перенаправляти ці дані в файл, на принтер або у порожній пристрій.

**Форматування виведеної інформації**

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

В оболонці **PowerShell** виведення форматують тільки чотири спеціальні командлети ***Format*** (табл. 8.1). Це спрощує вивчення, так як не потрібно запам'ятовувати засоби і параметри форматування для інших команд (інші командлети виведення не форматують).

Табл. 8.1. Командлети **PowerShell** для форматування виведення

|  |  |
| --- | --- |
| **Командлет** | **Опис** |
| ***Format-Table*** | Форматує виведення у вигляді таблиці, стовпці якої містять властивості об'єкта (також можуть бути додані обчислювані стовпці). Підтримується можливість угруповання даних, що виводяться |
| ***Format-List*** | Виводить об'єкт як список властивостей. При цьому кожна властивість відображається на новому рядку. Підтримується можливість угруповання даних, що виводяться |
| ***Format-Custom*** | Використовує для користувача представлення (view) для форматування виведення |
| ***Format-Wide*** | Форматує об'єкти у вигляді широкої таблиці, в якій відображується тільки одна властивість кожного об'єкта |

Якщо жоден з командлетів ***Format*** явно не вказаний, то використовується модуль форматування за замовчуванням, який визначається за типом даних, що відображаються. Наприклад, при виконанні командлету ***Get-Service*** дані за замовчуванням виводяться як таблиця з трьома стовпцями (***Status***, ***Name*** і ***DisplayName***):

***PS С:\> Get-Service***

***Status Name DisplayName***

***------ ---- -----------***

***Running AdobeARMservice Adobe Acrobat Update Service***

***Stopped AeLookupSvc Application Experience***

***Running AGSService Adobe Genuine Software Integrity …***

***Stopped ALG Application Layer Gateway Service***

***Stopped Apache2.2-Zend Apache2.2-Zend***

***Stopped AppIDSvc Application Identity***

***Running Appinfo Application Information***

***Stopped AppMgmt Application Management***

***Stopped aspnet\_state ASP.NET State Service***

***Stopped AudioEndpointBu... Windows Audio Endpoint Builder***

***Stopped AudioSrv Windows Audio***

Для зміни формату даних, що виводяться, потрібно направити їх по конвеєру відповідному командлету ***Format***. Наприклад, наступна команда виведе список служб за допомогою командлета ***Format-List***:

***PS С:\> Get-Service | Format-List***

***Name : AdobeARMservice***

***DisplayName : Adobe Acrobat Update Service***

***Status : Running***

***DependentServices : {}***

***ServicesDependedOn : {}***

***CanPauseAndContinue : False***

***CanShutdown : False***

***CanStop : True***

***ServiceType : Win32OwnProcess***

***Name : AeLookupSvc***

***DisplayName : Application Experience***

***Status : Stopped***

***DependentServices : {}***

***ServicesDependedOn : {}***

***CanPauseAndContinue : False***

***CanShutdown : False***

***CanStop : False***

***ServiceType : Win32ShareProcess***

***------------------------------------------------------------***

Як бачите, при використанні формату списку виводиться більше відомостей про кожну службу, ніж в форматі таблиці (замість трьох стовпців даних про кожну службі в форматі списку виводяться дев'ять рядків даних). Однак це зовсім не означає, що командлет ***Format-List*** витягує додаткові відомості про служби. Ці дані містяться в об'єктах, що повертаються командою ***Get-Service***, проте командлет, що використовується за замовчуванням, ***Format-Table*** відкидає їх, тому що не може вивести на екран більше трьох стовпців.

При форматуванні виведення за допомогою командлетів ***Format-List*** і ***Format-Table*** можна вказувати імена властивостей, які повинні бути відображені (нагадаємо, що переглянути список властивостей, наявних у об'єкта, дозволяє командлет ***Get-Member***). Наприклад:

***PS С:\> Get-Service | Format-List Name, Status, CanStop***

***Name : AdobeARMservice***

***Status : Running***

***CanStop : True***

***Name : AeLookupSvc***

***Status : Stopped***

***CanStop : False***

***---------------------***

Вивести всі наявні у об'єктів властивості можна за допомогою параметра ***\****. Наприклад:

***PS С:\> Get-Service | Format-List \****

**Перенаправлення інформації, що виводиться**

В оболонці **PowerShell** є кілька командлетів, за допомогою яких можна управляти виведенням даних. Ці командлети починаються зі слова ***Out***. Їх список можна побачити в такий спосіб:

***PS С:\> Get-Command Out-\* | Format-Table Name***

***Name***

***----***

***Out-Default***

***Out-File***

***Out-GridView***

***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 даного механізму потрібно враховувати, що архітектура **PowerShell** має на увазі відмінність між власне ядром оболонки (інтерпретатором команд) і головним додатком (host), який використовує це ядро. В принципі, в якості головного може виступати будь-який додаток, в якому реалізований ряд спеціальних інтерфейсів, що дозволяють коректно інтерпретувати отримувану від **PowerShell** інформацію. У нашому випадку головним додатком є ​​консольне вікно, в якому ми працюємо з оболонкою, і командлет ***Out-Host*** передає інформацію, що виводиться в це консольне вікно.

Параметр ***-Paging*** командлету ***Out-Host***, подібно команді ***more*** інтерпретатора **cmd.exe**, дозволяє організувати посторінковий вивід інформації, наприклад:

***PS С:\> Get-Help Get-Process -Full | Out-Host –Paging***

**Збереження даних в файл**

Як уже згадувалося раніше, **PowerShell** підтримує перенаправлення виведення команд в текстові файли за допомогою стандартних операторів ***>*** і ***>>***. Наприклад, наступна команда виведе вміст кореневого каталогу ***С:\*** в текстовий файл ***D:\dir\_c.txt*** (якщо такий файл існував, то він буде перезаписаний):

***PS С:\> dir С:\ > D:\dir\_c.txt***

Якщо потрібно перенаправити виведення команди в файл в режимі додавання (зі збереженням попереднього вмісту даного файлу), слід скористатися оператором ***>>***:

***PS С:\> dir С:\ >> D:\dir\_c.txt***

Крім операторів перенаправлення ***>*** і ***>>*** в **PowerShell** є командлет ***Out-File***, також дозволяє направити виведені дані замість вікна консолі в текстовий файл. При цьому командлет ***Out-File*** має кілька додаткових параметрів, за допомогою яких можна більш гнучко управляти виведенням: задавати тип кодування файлу, задавати довжину виведених рядків в знаках, вибирати режим перезапису файлу (табл. 8.4).

Табл. 8.4. Деякі параметри командлета ***Out-File***

|  |  |
| --- | --- |
| **Параметр** | **Опис** |
| ***-FilePath*** | Вказує шлях до вихідного файлу |
| ***-Encoding*** | Визначає кодування вихідного файлу. Можна вибрати зі значень **Unicode**, **UTF7**, **UTF8**, **UTF32**, **ASCII**, **BigEndianUnicode**, **Default** і **OEM**. За замовчуванням в **PowerShell** використовується кодування **Unicode**. Для збереження тексту в Windows-кодуванні слід вибирати значення **Default** (кодування поточної кодової сторінки ANSI), для збереження тексту в DOS-кодуванні - значення **OEM** |
| ***-Width*** | Вказує число знаків в кожній вихідному рядку |
| ***-Append*** | Записує вихідні дані в кінець існуючого файлу, а не заміщує його вміст |
| ***-NoClobber*** | Задає режим перезапису файлу. При вказівці цього параметра, якщо вихідний файл вже існує, він не буде перезаписуватись (за замовчуванням, якщо файл існує за вказаною шляху, командлет ***Out-File*** перезаписує його без попередження). Якщо одночасно використовуються параметри ***-Append*** і ***-NoClobber***, вихідні дані записуються в кінець існуючогофайлу |

Наприклад, наступна команда збереже в файлі ***C:\help.txt*** детальний варіант вбудованої довідки по командлету ***Get-Process*** (файл ***C:\help.txt*** буде створений в Windows-кодуванні):

***PS С:\> Get-Help Get-Process -Detailed | Out-File -FilePath C:\help.txt -Encoding "Default"***

**Друк даних**

Дані можна вивести безпосередньо на принтер за допомогою командлета ***Out-Printer***. При цьому друк може здійснюватися як на принтері за замовчуванням (ніяких спеціальних параметрів для цього вказувати не треба), так і на довільному принтері (в цьому випадку коротке ім'я принтера має бути зазначено в якості значення параметра ***-Name***). Наприклад:

***PS C:\script> Get-Process | Out-Printer -Name "Xerox Phaser 3500 PCL 6"***

**Придушення виведення**

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

***PS C:\> mkdir dir1***

***Directory: C:\***

***Mode LastWriteTime Length Name***

***---- ------------- ------ ----***

***d---- 21.03.2018 16:48 dir1***

Якщо ви не бажаєте бачити цю інформацію, то результат виконання команди ***mkdir*** потрібно передати по конвеєру командлету ***Out-Null***:

***PS С:\> mkdir klop | Out-Null***

У цьому випадку ніяких повідомлень на екран не виводиться.

***Змінні, масиви та хеш-таблиці***

У багатьох наведених раніше прикладах ми вже використали різні чіслові і символьні літерали (константи), а також змінні **PowerShell**, зберігаючи в них результати виконання команд. Крім змінних в **PowerShell**. як у багатьох інших мовах програмування, підтримуються масиви, а також більш специфічні структури - асоціативні масиви (хеш-таблиці). Розглянемо ці елементи мови **PowerShell** більш докладно.

Практично в кожній мові програмування є можливість роботи з числами і символьними рядками, причому способи їх завдання можуть бути. **PowerShell** також підтримує цілі і дійсні числа, а також символьні рядки кількох видів.

***Числові літерали***

Мова PowerShell підтримує всі основні числові типи платформи **.Net**: ***System.Int32***, ***System.Int64***, ***System.Double***. При цьому явно задавати тип чисел немає необхідності - система сама вибирає відповідний тип для зазначеного вами числа. Перевіримо тип декількох чисел, скориставшись для цього методом ***GetType***:

***PS С:\> (10).GetType().FullName***

***System.Int32***

***PS C:\> (10.23).GetType().FullName***

***System.Double***

***PS C:\> (10+10.23).GetType().FullName***

***System.Double***

У **PowerShell** передбачені спеціальні суфікси-множники для спрощення роботи з величинами, часто використовуваними системними адміністраторами: кілобайтами, мегабайтами і гігабайтами (табл. 9.1).

Табл. 9.1. Суфікси-множники в **PowerShell**

|  |  |  |  |
| --- | --- | --- | --- |
| **Суфікс-множник** | **Числовий множник** | **Приклад** | **Числове значення для прикладу** |
| ***КВ*** | ***1024*** | ***2 КВ*** | ***2048*** |
| ***kb*** | ***1024*** | ***1 kb*** | ***1126.4*** |
| ***МВ*** | ***1024\*1024*** | ***3MB*** | ***3 145 728*** |
| ***mb*** | ***1024\*1024*** | ***2.5mb*** | ***2 621 440*** |
| ***GB*** | ***1024\*1024\*1024*** | ***1GB*** | ***1 073 741 824*** |
| ***gb*** | ***1024\*1024\*1024*** | ***2.23gb*** | ***2 394 444 267.52*** |

Наведемо приклади:

***PS С:\> 1mb+10kb***

***1058816***

***PS С:\> 2GB+56MB***

***2206203904***

У **PowerShell** можна оперувати числами в шістнадцятковому форматі, використовуючи для цього ті ж позначення, що і в С-подібних мовах програмування: перед числом вказується префікс ***0х***, а в запису числа можуть бути присутніми цифри і букви ***А***, ***B***, ***C***, ***D***, ***E*** і ***F*** (незалежно від регістру). Наприклад:

***PS С:\> 0x10***

***16***

***PS С:\> 0хА***

***10***

***PS С:\> 0xcd***

***205***

***Символьні рядки***

Всі символьні рядки в **PowerShell** є об'єктами типу ***System.String*** і являють собою послідовність 32-бітових символів в кодуванні **Unicode**. Довжина рядків не обмежена, вміст рядків не можна змінювати (можна тільки копіювати).

У **PowerShell** підтримується чотири види символьних рядків.

Рядки можуть задаватися послідовністю символів, укладених в одинарні або подвійні лапки:

***PS С:\> 'Рядок в одинарних лапках'***

***Рядок в одинарних лапках***

***PS С:\> "Рядок в подвійних лапках"***

***Рядок в подвійних лапках***

Рядки можуть містити будь-які символи (в тому числі символи розриву рядка і повернення каретки), крім відповідного одиночного закриваючого символу (одинарної або подвійної лапки). Рядок в одинарних лапках може містити подвійні лапки і навпаки:

***PS С:\> 'Рядок в "одинарних" лапках'***

***Рядок в "одинарних" лапках***

***PS С:\> "Рядок в 'подвійних' лапках"***

***Рядок в 'подвійних' лапках***

Якщо всередині рядка потрібно помістити символ, що обмежує даний рядок (тобто одинарні або подвійні лапки), то потрібно написати цей символ два рази поспіль:

***PS С:\> 'Рядок в ''одинарних лапках'***

***Рядок в 'одинарних лапках***

***PS С:\> "Рядок в ""подвійних лапках"***

***Рядок в "подвійних лапках***

Рядки в подвійних лапках є розширюваними. Це означає, що якщо всередині рядка в подвійних лапках зустрічається ім'я змінної або інший вираз, який може бути обчислений, то в даний рядок підставляється значення даної змінної або результат обчислення виразу. Наприклад:

***PS С:\> $а = 123***

***PS С:\> "$а дорівнює $а"***

***123 дорівнює 123***

Якщо ім'я змінної зустрічається всередині рядка в одинарних лапках, то ніякої підстановки значення змінної не відбувається:

***PS С:\> '$а дорівнює $а'***

***$a дорівнює $а***

При необхідності можна відключити розширення певної змінної всередині рядка в подвійних лапках. Для цього перед знаком ***$*** цій змінній потрібно вказати символ зворотного апострофа (***`***) наприклад:

***PS С:\> "`$а дорівнює $а"***

***$а дорівнює 123***

Символи, що мають спеціальне значення, вставляються в рядки в подвійних лапках за допомогою escape-послідовностей, які в **PowerShell** починаються з символу зворотного апострофа (***`***).В інших мовах програмування типу **С**, ***С#***, ***JScript*** або ***Perl*** для виділення спеціальних символів (escape-послідовностей) використовується зворотна коса риска (наприклад, ***\n*** або ***\t***). Розробники оболонки **PowerShell** прийняли рішення ввести інший символ для escape-послідовностей, щоб уникнути проблем при використанні символу ***\*** як розділювача компонентів шляху в файловій системі **Windows** та інших просторах імен **PowerShell**.

Вставимо символ розриву рядка в рядок в подвійних лапках:

***PS С:\> "Рядок в `nподвійних лапках"***

***Рядок в***

***подвійних лапках***

Як бачите, на екран інформація виводиться в двох рядках. Якщо ж вставити escape-послідовність ***`n*** в рядок в одинарних лапках, то розриву рядка не відбудеться:

***PS С:\> 'Рядок в `nодинарних лапках'***

***Рядок в `nодинарних лапках***

Крім змінних в розширюваних рядках можуть зазначатися так звані підвирази (subexpression) - обмежені символами ***$(...)*** фрагменти коду на мові **PowerShell**, які в рядках замінюються на результати обчислення цих фрагментів. Наприклад:

***PS С:\> "3+2 дорівнює $(3+2)"***

***3+2 дорівнює 5***

***Автономні рядки***

У **PowerShell** поряд зі звичайними рядками в одинарних і подвійних лапках підтримуються так звані автономні рядки (також відомі під ім'ям рядків типу "here-string"). Подібні рядки зазвичай використовуються для вставки в сценарій великих блоків тексту або при генерації текстової інформаціі для інших програм і мають такий вигляд:

***@ <лапка> <розрив\_рядка> блок тексту <розрив\_рядка><лапка>@***

Лапки можуть бути як одинарними, так і подвійними, при цьому сенс їх залишається тим же, що і для звичайних рядків: змінні і підвирази, які стоять всередині подвійних лапок, замінюються їх значеннями, а які стоять всередині одинарних лапок залишаються незмінними. Наприклад:

***PS С:\> $а=@"***

***>> 1 Перший рядок***

***>> $(1+1) Другий рядок***

***>> "Третій рядок"***

***>> "@***

***>>***

***PS C:\> $а***

***1 Перший рядок***

***2 Другий рядок***

***"Третій рядок"***

***PS C:\> $а=@'***

***>> 1 Перший рядок***

***>> $(1+1) Другий рядок***

***>> 'Третій рядок'***

***>> '@***

***>>***

***PS C:\> $а***

***1 Перший рядок***

***$(1+1) Другий рядок***

***'Третій рядок'***

***Змінні PowerShell***

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

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

Перевірити, чи визначена змінна, можна за допомогою командлета ***Test-Path***. Наприклад, наступна команда перевіряє, чи існує змінна ***MyVariable***:

***PS С:\> Test-Path Variable:MyVariable***

***False***

Список всіх змінних, визначених у поточному сеансі роботи, можна побачити, звернувшись до віртуального диску **PowerShell** ***Variable:*** за допомогою команди ***dir***:

***PS С:\> dir Variable:***

***Name Value***

***---- -----***

***$ Variable:MyVariable***

***? True***

***^ Test-Path***

***\_***

***a 1 Перший рядок...***

***args {}***

***ConfirmPreference High***

***ConsoleFileName***

***DebugPreference SilentlyContinue***

***---------------------------------------------------------***

***Змінні оболонки PowerShell***

**Змінні оболонки** - це набір змінних, які створюються, оголошуються оболонкою **PowerShell** і присутні за замовчуванням в кожному сеансі роботи. Змінні оболонки зберігаються протягом всього сеансу і доступні всім командам, сценаріями і додатків, які виконуються в даному сеансі.

Підтримуються два види змінних оболонки.

□ **Автоматичні змінні**. У цих змінних зберігаються параметри стану оболонки **PowerShell**. Автоматичні змінні зберігаються і динамічно змінюються самою системою. Його користувачі не можуть (і не повинні) змінювати значення цих змінних. Наприклад, значенням змінної $ PID є ідентифікатор поточного процесу PowerShell.exe.

□ **Змінні налаштувань**. У цих змінних зберігаються настройки активного користувача. Ці змінні створюються оболонкою **PowerShell** і заповнюються значеннями за замовчуванням. Користувачі можуть змінювати значення цих змінних. Наприклад, змінна ***$MaximumHistoryCount*** визначає максимальне число записів в журналі сеансу.

У табл. 9.3 наведено короткий опис змінних оболонки.

Табл. 9.3. Змінні оболонки **PowerShell**

|  |  |
| --- | --- |
| **Змінна** | **Опис** |
| ***$$*** | Містить останню лексему останнього отриманого оболонкою рядка |
| ***$?*** | Показує, чи успішно завершилася остання операція |
| ***$^*** | Містить першу лексему останнього отриманого оболонкою рядка |
| ***$\_*** | При використанні в блоках сценаріїв, фільтрах і інструкції ***Where*** містить поточний об'єкт конвеєра |
| ***$Args*** | Містить масив параметрів, що передаються у функцію |
| ***$Error*** | Містить об'єкти, для яких виникла помилка при обробці в командлеті |
| ***$ForEach*** | Звертається до ітератору в циклі ***ForEach*** |
| ***$Home*** | Вказує домашній каталог користувача. Еквівалент конструкції ***%HomeDrive%%HomePath%*** в оболонці **cmd.exe** |
| ***$Input*** | Використовується в блоках сценаріїв, що знаходяться в конвеєрі |
| ***$PSHome*** | Показує ім'я каталогу, в якому встановлений **PowerShell** |
| ***$Host*** | Містить відомості про поточний вузол |
| ***$OFS*** | Використовується як розділювач при перетворенні масиву в рядок. За замовчуванням ця змінна має значення проміжку |

Змінними оболонки можна користуватися так само, як і іншими видами змінних. Наприклад, наступна команда виведе на екран вміст домашнього каталогу **PowerShell**, шлях до якого зберігається в змінній оболонки ***$PSHome***:

***PS С:\> dir $PSHome***

***Directory: C:\Windows\System32\WindowsPowerShell\v1.0***

***Mode LastWriteTime Length Name***

***---- ------------- ------ ----***

***d---- 21.11.2010 9:59 en-US***

***d---- 14.07.2009 8:37 Examples***

***d---- 14.07.2009 8:37 Modules***

***-a--- 10.06.2009 23:41 27338 Certificate.format.ps1xml***

***-a--- 10.06.2009 23:41 27106 Diagnostics.Format.ps1xml***

***-a--- 10.06.2009 23:41 72654 DotNetTypes.format.ps1xml***

***-a--- 10.06.2009 23:41 24857 FileSystem.format.ps1xml***

***-a--- 10.06.2009 23:42 15603 getevent.types.ps1xml***

***-a--- 10.06.2009 23:41 257847 Help.format.ps1xml***

***----------------------------------------------------------------------***

***Користувацькі змінні. Типи змінних***

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

***PS С:\> $а=1***

***PS С:\> $а***

***1***

***PS С:\> Test-Path Variable:а***

***True***

***PS С:\> dir Variable:а***

***Name Value***

***---- -----***

***a 1***

Перевіримо, який тип має змінна ***$а***. Для цього можна скористатися командлетом ***Get-Member*** або методом ***GetType()***".

***PS С:\> $а | Get-Member***

***TypeName: System.Int32***

***----------------------------***

***PS C:\> $a.GetType().FullName***

***System.Int32***

Отже, змінна ***$a*** зараз має тип ***System.Int32***. Дамо цій зміннійй інше значення (рядок) і знову перевіримо тип:

***PS С:\> $а="ааа"***

***PS С:\> $а | Get-Member***

***TypeName: System.String***

***-----------------------***

Можна також явно вказати тип змінної при її визначенні, вказавши в квадратних дужках відповідний атрибут типу. При цьому вираз, що стоїть в правій частині після знака рівності, буде перетворено (якщо це можливо) до даного типу. Наприклад, оголосимо цілочисельну змінну ***$а*** й дамо цій змінної символьне значення, яке можна перетворити до цілого типу:

***PS С:\> [System.Int32]$a=10***

***PS С:\> $а="123"***

***PS С:\> $а***

***123***

***PS С:\> $a.GetType().FullName***

***System.Int32***

Як бачите, рядок "***123"*** був перетворений в ціле число ***123***. Якщо ж спробувати записати в змінну ***$а*** значення, яке не може бути перетворено в ціле число, то виникне помилка:

***PS С:\> $а="ааа"***

***Cannot convert value "aaa" to type "System.Int32". Error: "Input string was not in a correct format."***

***At line:1 char:3***

***--------------------------------------------------------------***

Замість явного вказівки .NET-типу змінної можна користуватися більш короткими псевдонімами типів відповідно до мови С#. Наприклад:

***PS С:\> [int]$а=10***

***PS С:\> $a.GetType().FullName***

***System.Int32***

***Масиви у PowerShell***

На відміну від багатьох мов програмування, в **PowerShell** не потрібно за допомогою будь-яких спеціальних символів вказувати початок масиву або його кінець, а також попередньо оголошувати масив.

Для створення і ініціалізації масиву можна просто привласнити значення його елементів. Значення, що додаються в масив, розділяються комою і відділяються від імені змінної (імені масиву) оператором присвоювання. Наприклад, наступна команда створить масив ***$а*** з трьох елементів:

***PS С:\> $а=1,2,3***

***PS С:\> $а***

***1***

***2***

***3***

Можна також створити і ініціалізувати масив, використовуючи оператор діапазону ***(..)***. Наприклад, щоб створити і ініціалізувати масив ***$b***, що містить значення від ***10*** до ***14***, можна виконати наступну команду:

***PS С:\> $b=10..14***

В результаті масив ***$b*** буде містити п'ять значень:

***PS С:\> $b***

***10***

***11***

***12***

***13***

***14***

***Звернення до елементів масиву***

Як ми вже бачили, для відображення всіх елементів масиву потрібно просто ввести його ім'я.

Довжина масиву (кількість елементів) зберігається у властивості ***Length***:

***PS С:\> $а.Length***

***3***

Для звернення до певного елементу масиву потрібно вказати його порядковий номер (індекс) в квадратних дужках після імені змінної. При цьому слід мати на увазі, що нумерація елементів в масиві завжди починається з нуля, тому для отримання значення першого елемента потрібно виконати наступну команду:

***PS С:\> $а[0]***

***1***

В якості індексу можна вказувати і негативні значення, при цьому відлік буде вестися з кінця масиву. Наприклад, індекс ***-1*** відповідатиме останньому елементу масиву:

***PS С:\> $а[-1]***

***3***

Щоб відобразити підмножину всіх значень в масиві, можна застосовувати оператор діапазону. Наприклад, щоб відобразити елементи з індексами від ***1*** до ***2***, потрібно ввести:

***PS С:\> $а[1..2]***

***2***

***3***

В операторі діапазону можна використовувати властивість ***Length***. Наприклад, для відображення елементів від індексу ***1*** до кінця масиву (останній елемент масиву має індекс ***Length-1***) можна виконати наступну команду:

***PS С:\> $а[1..($а.Length-1)]***

***2***

***3***

Для зміни елемента масиву потрібно присвоїти нове значення елементу з відповідним індексом:

***PS С:\> $а[0]=5***

***PS С:\> $а[1]=3.14***

***PS С:\> $а[2]="привіт"***

***PS С:\> $а***

***5***

***3.14***

***Привіт***

***Операції з масивом***

Останній приклад показує, що за замовчуванням масиви **PowerShell** можуть містити елементи різних типів, тобто є поліморфними. Визначимо тип нашого масиву ***$а***:

***PS С:\> $а.GetType().FullName***

***System.Object[]***

Отже, змінна ***$а*** має тип "масив елементів типу ***System.Object***". Можна створити масив з жорстко заданим типом, тобто такий масив, який містить елементи тільки одного типу. Для цього, як і у випадку зі звичайними скалярними змінними, необхідно вказати потрібний тип в квадратних дужках перед іменем змінної. Наприклад, наступна команда створить масив 32-розрядних цілих чисел:

***PS С:\> [int[]]$а=1,2,3,4***

Якщо спробувати записати в даний масив значення, яке не можна перетворити до цілого типу, то виникне помилка:

***PS С:\> $а[0]="ааа"***

***Array assignment to [0] failed: Cannot convert value "aaa" to type "System.Int32". Error: "Input string was not in a correct format.".***

***At line:1 char:4***

***--------------------------------------------------------------***

При спробі звернутися до елементу, який виходить за межі масиву, виникне помилка. Наприклад:

***PS С:\> $а.Length***

***4***

***PS С:\> $а[4]=5***

***Array assignment failed because index '4' was out of range.***

***At line:1 char:4***

***--------------------------------------------------------------***

Подібні помилки пов'язані з тим. що масиви **PowerShell** базуються на .NET-масивах, що мають фіксовану довжину. Незважаючи на це. є спосіб збільшення довжини масиву. Для цього можна скористатися оператором конкатенації ***+*** або ***+=***. Наприклад, наступна команда додасть до масиву ***$а*** два нові елементи зі значеннями ***5*** і ***6***:

***PS С:\> $а***

***1***

***2***

***3***

***4***

***PS С:\> $а+=5,6***

***PS С:\> $а***

***1***

***2***

***3***

***4***

***5***

***6***

При виконанні оператора ***+=*** відбувається наступне:

1. **PowerShell** створює новий масив, розмір якого достатній для поміщення в нього всіх елементів;
2. Первісний вміст масиву копіюється в новий масив;
3. Нові елементи копіюються в кінець нового масиву.

Таким чином, ми насправді не додаємо новий елемент до масиву, а створюємо новий масив більшого розміру.

Видалити елемент з масиву не так просто, проте можна створити новий масив і скопіювати в нього всі елементи крім непотрібного. Наприклад, наступна команда створить масив ***$b***, що містить всі елементи масиву крім значення з індексом ***2***:

***PS С:\> $b = $а[0,1 + 3..($а.Length-1)]***

***PS С:\> $b***

***1***

***2***

***4***

***5***

***6***

Можна об'єднати два масиви в один за допомогою оператора конкатенації ***+***. Наприклад:

***PS С:\> $х=1,2***

***PS С:\> $у=3,4***

***PS С:\> $z=$x+$y***

***PS С:\> $z***

***1***

***2***

***3***

***4***

Слід мати на увазі, що звичайний оператор присвоювання (***=***) діє на масиви за посиланням. Наприклад, створимо масив ***$а*** з двох елементів і присвоїмо цей масив змінній ***$b***:

***PS С:\> $а=1,2***

***PS С:\> $b=$а***

***PS С:\> $b***

***1***

***2***

Тепер змінимо значення першого елемента масиву ***$а*** й подивимося, що відбудеться з масивом ***$b***:

***PS С:\> $а[0]="Нове значення"***

***PS С:\> $b***

***Нове значення***

***2***

Як бачите, вміст масиву ***$b*** також змінився, так як змінна $b вказує на той же об'єкт, що і змінна ***$а***.

Для видалення масиву можна скористатися командлетом ***Remove-Item*** (псевдонім ***del***) і видалити змінну, яка містить потрібний масив, з віртуального диску ***variable:***. Наприклад:

***PS C:\> $a***

***Нове значення***

***2***

***PS C:\> del variable:a***

***PS C:\> $a***

***Хеш-таблиці (асоціативні масиви)***

Крім звичайних масивів в **PowerShell** підтримуються так звані **асоціативні масиви** (іноді їх також називають **словниками**) - структури для зберігання колекції ключів і їх значень, пов'язаних попарно.

Наприклад, можна використовувати прізвище людини як ключ, а його дату народження як значення. Асоціативний масив забезпечує структуру для зберігання колекції імен і дат народження, де кожному імені зіставлена ​​дата народження. Візуально масив асоційованих значень можна уявити як таблицю, що складається з двох стовпців, де перший стовпець є ключем, а другий - значенням.

Асоціативні масиви схожі на звичайні масиви в **PowerShell**, але замість звернення до вмісту масиву за індексом, до елементів даних асоціативного масиву можна звертатися по ключу. Використовуючи цей ключ. **PowerShell** повертає відповідне значення з асоціативного масиву.

Для зберігання вмісту асоціативного масиву в **PowerShell** використовується спеціальний тип даних - хеш-таблиця, оскільки дана структура даних забезпечує швидкий механізм пошуку. Це дуже важливо, оскільки основним призначенням асоціативного масиву є забезпечення ефективного механізму пошуку.

На відміну від звичайного масиву для оголошення і ініціалізації хеш-таблиць використовуються спеціальні літерали:

***$ім’я\_масиву = @{ключ1 = елемент1; ключ2 = елемент2; ...}***

Отже, кожному значенню хеш-таблиці потрібно прикріпити мітку (ключ), перед перерахуванням вмісту масиву потрібно поставити символи ***@{***, а завершити перерахування елементів символом ***}***. Ключі та значення поділяються знаком рівності (***=***). пари "ключ-значення" розділяються між собою крапкою з комою (***;***).

Створимо, наприклад, хеш-таблицю, в якій будуть зберігатися дані про одну людину (асоціативний масив з трьома елементами):

***PS C:\> $user=@{Surname="Іванов";Name="Іван";Phone="11-11-11"}***

***PS C:\> $user***

***Name Value***

***---- -----***

***Name Іван***

***Phone 11-11-11***

***Surname Іванов***

Тепер потрібно навчитися звертатись до її елементів. У **PowerShell** доступ до хеш-таблиць можливий двома способами: з використанням нотації властивостей або нотації масивів. Звернення з використанням нотації властивостей виглядає наступним чином:

***PS C:\> $user.Surname***

***Іванов***

***PS C:\> $user.Name***

***Іван***

При цьому підході хеш-таблиця розглядається як об'єкт: ви вказуєте ім'я потрібного властивості і отримуєте відповідне значення. Звернення до асоціативного масиву з використанням нотації масивів відбувається так:

***PS C:\> $user["Surname"]***

***Іванов***

***PS C:\> $user["Name"]***

***Іван***

***PS C:\> $user["Surname","Name"]***

***Іванов***

***Іван***

При роботі з хеш-таблицею як з масивом можна отримувати значення відразу для декількох ключів.

Базовим типом для асоціативних масивів PowerShell є тип ***System.Collections.Hashtable***:

***PS C:\> $user.GetType().FullName***

***System.Collections.Hashtable***

В даному типі визначені кілька властивостей і методів, які можна використовувати (нагадаємо, що повний список властивостей і методів можна отримати з допомогою командлету ***Get-Member***). Наприклад, у властивостях ***Keys*** і ***Values*** зберігаються всі ключі і все значення, відповідно:

***PS C:\> $user.Keys***

***Name***

***Phone***

***Surname***

***PS C:\> $user.Values***

***Іван***

***11-11-11***

***Іванов***

***Операції з хеш-таблицею***

Давайте навчимося додавати елементи в хеш-таблицю, змінювати і видаляти їх. Додамо в хеш-таблицю ***$user*** дані про вік людини і про місто, де вона проживає:

***PS C:\> $user.Age=30***

***PS C:\> $user***

***Name Value***

***---- -----***

***Name Іван***

***Age 30***

***Phone 11-11-11***

***Surname Іванов***

***PS C:\> $user["City"]="Миколаїв"***

***PS C:\> $user***

***Name Value***

***---- -----***

***City Миколаїв***

***Name Іван***

***Age 30***

***Phone 11-11-11***

***Surname Іванов***

Таким чином, додаються елементи в асоціативний масив за допомогою простого оператора присвоювання з використанням нотації властивостей або масивів. Тепер змінимо значення вже наявного в масиві ключа. Робиться це також за допомогою оператора присвоювання:

***PS C:\> $user["City"]="Херсон"***

***PS C:\> $user***

***Name Value***

***---- -----***

***City Херсон***

***Surname Іванов***

***Name Іван***

***Age 30***

***Phone 11-11-11***

Для видалення елемента з асоціативного масиву використовується метод ***Remove()***:

***PS C:\> $user.Remove("Age")***

***PS C:\> $user***

***Name Value***

***---- -----***

***City Херсон***

***Surname Іванов***

***Name Іван***

***Phone 11-11-11***

Можна створити порожню хеш-таблицю, що не вказуючи жодної пари "ключ-значення". І потім заповнювати її послідовно по одному елементу:

***PS C:\> $a=@{}***

***PS С:\> $а***

***PS С:\> $а.one=1***

***PS С:\> $a.two=2***

***PS С:\> $a***

***Name Value***

***---- -----***

***one 1***

***two 2***

Як і у випадку зі звичайними масивами, оператор присвоювання діє на хеш-таблиці за посиланням. Наприклад, після виконання наступних команд змінні ***$а*** і ***$b*** вказуватимуть на один і той же об'єкт:

***PS C:\> $b=$a***

***PS C:\> $b***

***Name Value***

***---- -----***

***one 1***

***two 2***

Змінивши значення одного з елементів в ***$a***, ми отримаємо той же зміна в ***$b***:

***PS C:\> $a.one=3***

***PS C:\> $b***

***Name Value***

***---- -----***

***one 3***

***two 2***

**Лекція №9. «Керуючі конструкції мови програмування PowerShell. Написання функцій та сценаріїв у PowerShell. Використання об’єктів .NET та зовнішніх об’єктів у PowerShell»**

***Оператори і керуючі інструкції***

У мові **PowerShell** підтримується багато операторів, що дозволяють виконувати різні дії. Однією з особливостей операторів **PowerShell** є їх поліморфізм, тобто можливість застосовувати один і той же оператор до об'єктів різних типів. При цьому відмінність **PowerShell** від багатьох інших об'єктно-орієнтованих мов програмування полягає в тому, що поведінка операторів для основних типів даних (рядки, числа, масиви і хеш-таблиці) реалізується безпосередньо інтерпретатором, а не за допомогою того чи іншого методу об'єктів.

***Арифметичні оператори***

Основні арифметичні оператори, які підтримуються в **PowerShell**, приведені в табл. 9.1.

Табл. 9.1. Основні арифметичні оператори в **PowerShell**

|  |  |  |  |
| --- | --- | --- | --- |
| **Оператор** | **Опис** | **Приклад** | **Результат** |
| ***+*** | Складає два значення | ***2+4***  ***"ааа"+"bbb"***  ***1,2,3+4,5*** | ***6***  ***"aaabbb"***  ***1,2,3,4,5*** |
| ***\**** | Перемножує два значення | ***2\*4***  ***"а"\*3***  ***1,2,3\*2*** | ***8***  ***"ааа"***  ***1,2,3,1,2,3*** |
| ***-*** | Віднімає одне значення з іншого | ***5-3*** | ***2*** |
| ***/*** | Ділить одне значення на інше | ***6/3***  ***7/4*** | ***2***  ***1.75*** |
| ***%*** | Повертає залишок при цілочисельному діленні одного значення на інше | ***7%4*** | ***3*** |

З точки зору поліморфної поведінки найбільш цікавими є оператори додавання і множення. Розглянемо їх більш детально.

***Оператор додавання***

Як уже згадувалося, поведінка операторів ***+*** і ***\**** для чисел, рядків, масивів і хеш-таблиць визначається самим інтерпретатором **PowerShell**. У результаті додавання або множення двох чисел виходить число. Результатом додавання (конкатенації) двох рядків є рядок. При додаванні двох масивів створюється новий масив, який є об'єднанням масивів.

А що станеться, якщо спробувати скласти об'єкти різних типів (наприклад, число з рядком)? У цьому випадку поведінка оператора буде визначатись типом операнда, що стоїть зліва. Слід запам'ятати так зване "правило лівої руки": тип операнда, що стоїть зліва, задає тип результату дії оператора.

Якщо лівий операнд є числом, то **PowerShell** спробує перетворити правий операнд до числового типу. Наприклад:

***PS C:\> 1+"12"***

***13***

Як бачите, рядок ***"12"*** була перетворена до числа ***12***. Результат дії оператора додавання – число ***13***. Тепер зворотний приклад, коли лівий операнд є рядком:

***PS C:\> "1"+12***

***112***

Тут число ***12*** перетворюється до рядка ***"12"*** і в результаті конкатенації повертається рядок ***"112"***.

Якщо правий операнд можна перетворити до типу лівого операнда, то виникне помилка:

***PS C:\> 1+"a"***

***Не удается преобразовать значение "a" в тип "System.Int32". Ошибка: "Входная строка имела неверный формат."***

***строка:1 знак:1***

***+ 1+"a"***

Якщо операнд, що стоїть зліва від оператора складання, є масивом, то стоїть праворуч операнд додається до цього масиву. При цьому створюється новий масив типу (***object []]***, в який копіюється вміст операндів (це пов'язано з тим, що розмірність .NET-масивів фіксована). В процесі створення нового масиву всі обмеження на типи масивів, що додаються, будуть втрачені. Розглянемо приклад. Створимо спочатку масив цілих чисел:

***PS C:\> $а = [int[]] (1,2,3,4)***

***PS C:\> $a.GetType().FullName***

***System.Int32[]***

Якщо спробувати змінити значення елемента цього масиву на який-небудь рядок, то виникне помилка, так як елементами масиву ***$а*** можуть бути тільки цілі числа:

***PS C:\> $a[0]="aaa"***

***Не удается преобразовать значение "aaa" в тип "System.Int32". Ошибка: "Входная строка имела неверный формат."***

***строка:1 знак:1***

***+ $a[0]="aaa"***

Додамо тепер до масиву ***$а*** ще один символьний елемент за допомогою оператора додавання:

***PS C:\> $a=$a+"abc"***

Знову спробуємо змінити значення першого елемента масиву ***$а***, записавши в нього символьний рядок:

***PS C:\> $a[0]="aaa"***

***PS C:\> $a***

***aaa***

***2***

***3***

***4***

***abc***

Як бачите, тепер помилка не виникає. Це пов'язано зі зміною типу масиву ***$а***:

***PS C:\> $a.GetType().FullName***

***System.Object[]***

Після збільшення масив ***$а*** отримує тип ***[object[]]***, і його елементами можуть бути об'єкти будь-якого типу.

Перейдемо тепер до складання хеш-таблиць. Як і у випадку зі звичайними масивами, при додаванні хеш-таблиць створюється новий асоціативний масив, в який копіюються елементи з масивів, що складаються. При цьому обидва операнди повинні бути хеш-таблицями (ніяке перетворення типів тут не підтримується). У новий масив спочатку копіюються елементи хеш-таблиці, що стоїть зліва від оператора складання, а потім елементи хеш-таблиці, що стоїть праворуч. Якщо ключове значення з правого операнда вже зустрічалося в лівому, то при додаванні виникне помилка.

Розглянемо приклад:

***PS C:\> $left=@{a=1;b=2;c=3}***

***PS C:\> $right=@{d=4;e=5}***

***PS C:\> $sum=$left+$right***

***PS C:\> $sum***

***Name Value***

***---- -----***

***c 3***

***e 5***

***a 1***

***d 4***

***b 2***

Нова результуюча хеш-таблиця як і раніше має тип ***System.Collections.Hashtable***:

***PS C:\> $sum.GetType().FullName***

***System.Collections.Hashtable***

***Оператор множення***

Оператор множення може діяти на числа, рядки і звичайні массі¬ви (операція множення хеш-таблиць не визначена). При цьому праворуч від оператора множення обов'язково повинно знаходитися число (в іншому випадку виникне помилка).

Якщо зліва від оператора множення розташоване число, то операція множення виконується звичайним чином:

***PS C:\> 6\*5***

***30***

Якщо лівим операндом є рядок, то він повторюється вказану кількість разів:

***PS С:\> "abc"\*3***

***abcabcabc***

Аналогічним чином оператор множення діє на масиви:

***PS C:\> $а=1,2,3***

***PS C:\> $а=$а\*2***

***PS C:\> $а***

***1***

***2***

***3***

***1***

***2***

***3***

Як і в випадку оператора складання, при множенні масиву на число створюється новий масив типу ***[object[]]*** потрібного розміру і в нього копіюються елементи початкового масиву.

***Оператори віднімання, ділення і залишку від ділення***

Оператори віднімання (***-***), ділення (***/***) і залишку відділення (***%***) в **PowerShell** визначені тільки для чисел. Спеціального оператора цілочисельного ділення не передбачено: якщо діляться два цілих числа, то результатом буде дійсне число (тип ***System.Double***). наприклад:

***PS C:\> 123/4***

***30.75***

Якщо потрібно округлити результат до найближчого цілого числа, то слід просто перетворити його до типу ***[int]***. Наприклад:

***PS C:\> [int](123/4)***

***31***

При цьому слід мати на увазі, що **PowerShell** при перетворенні дійсного числа в ціле використовує так зване "округлення Банкера": якщо число є напівцілим (***0.5***, ***1.5***, ***2.5*** і т. д.). то воно округлюється до найближчого парного числа. Таким чином, числа ***1.5*** і ***2.5*** округлюються до ***2***. а ***3.5*** і ***4.5*** округлюються до ***4***.

Якщо один з операндів є числом, а другий ні, то **PowerShell** намагається виконати перетворення до числовому типу. наприклад:

***PS C:\> "123"/4***

***30.75***

***PS С:\> 123/"4"***

***30.75***

***Оператор присвоювання***

Поряд з простим оператором присвоювання (***=***) в PowerShell підтримуються С-подібні складені оператори присвоювання: ***+=***, ***-=***, ***\*=***, ***/=***, ***%=***.

***Оператори перевірки на відповідність шаблону***

Поряд з основними операторами порівняння в **PowerShell** є оператори перевірки символьних рядків на відповідність певним шаблоном. При цьому підтримуються два види шаблонів: **вирази з груповими символами** і **регулярні вирази**.

***Шаблони з груповими символами***

Раніше ми вже використали підстановочні (шаблонні) символи (. і ***?***) при використанні, скажімо, командлету ***dir***. Наприклад, команда ***dir \*.doc*** виводить всі файли в поточному каталозі, що мають розширення ***doc***, - і будь-яке ім'я (шаблонний символ ***\**** замінює будь-яку кількість будь-яких символів).

У **PowerShell** підтримуються чотири види групових символів (табл. 9.2) і кілька операторів, які перевіряють рядки на відповідність шаблонами з підстановочними символами (були розглянуті вище).

Табл. 9.2. Групові символи в **PowerShell**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Символ** | **Опис** | **Приклад** | **Відповідає** | **Не відповідає** |
| ***\**** | Будь-яка кількість довільних символів | ***a\**** | ***а***  ***аb***  ***abc*** | ***bс***  ***babe*** |
| ***?*** | Один довільний символ | ***a?b*** | ***асb***  ***a1b*** | ***а***  ***аb*** |
| ***[<симв 1>***  ***-<симв2>]*** | Діапазон символів від ***<симв1>*** до ***<симв2>*** | ***a[b-d]c*** | ***abc***  ***асc*** | ***аас***  ***ас*** |
| ***[<симв1>***  ***<симв2>...]*** | Будь-який символ зі вказаного набору | ***a[bc]c*** | ***abc***  ***асе*** | ***а***  ***adc*** |

***Шаблони з регулярними виразами***

Регулярні вирази узагальнюють і розширюють концепцію шаблонів з груповими символами. Для завдання зразка використовуються літерали і метасимволи. Кожен символ, який не має спеціального значення в регулярних виразах, розглядається як буквальний і повинен точно збігатися при пошуку. Наприклад, букви і цифри є літеральнимі символами. Метасимволи - це символи зі спеціальним значенням у регулярних виразах (табл. 9.3).

Табл. 9.3. Деякі метасимволи для регулярних виразів в **PowerShell**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Символ** | **Опис** | **Приклад** | **Відповідає** | **Не відповідає** |
| ***.*** | Символ підстановки: відповідає будь-якому символу | ***a..*** | ***abc***  ***a34*** | ***a***  ***ab*** |
| ***\**** | Повторювач: означає нуль або більше попередніх символів або класів символів | ***a.\*b*** | ***ab***  ***abc***  ***awerbc*** | ***a***  ***bbc*** |
| ***?*** | Повторювач: означає нуль або один попередній символ або клас символів | ***a?b*** | ***b***  ***ab***  ***cb*** | ***ac***  ***da*** |
| ***^*** | Положення в рядку: позначає початок рядка | ***^ab*** | ***abc***  ***abcd*** | ***sab***  ***acb*** |
| ***$*** | Положення в рядку: позначає кінець рядка | ***ab$*** | ***cdab***  ***ab*** | ***abc***  ***abb*** |
| ***[<симв 1>***  ***<симв2>…]*** | Клас символів: позначає будь-який символ з вказаної множини | ***a[bc]c*** | ***abc***  ***acc*** | ***a***  ***adc*** |
| ***[^<симв 1>***  ***<симв2>]*** | Інвертований клас: позначає будь-який символ, який не входить у вказану множину | ***a[^bc]c*** | ***adc***  ***a1c*** | ***abc***  ***acc*** |
| ***[<симв 1>-***  ***<симв2>]*** | Діапазон: позначає будь-який символ з вказаного проміжку | ***a[b-d]c*** | ***abc***  ***acc*** | ***aac***  ***ac*** |
| ***\<метасимвол>*** | Виняток: визначає використання метасимвола як литерала | ***\^ab*** | ***c^ab***  ***^ab*** | ***ab***  ***^ac*** |

У **PowerShell** з регулярними виразами працюють оператори ***-match***, ***-notmatch*** і ***-replace***, а також їх варіанти.

Табл. 9.4. Оператори, що працюють з регулярними виразами

|  |  |  |  |
| --- | --- | --- | --- |
| **Оператор** | **Опис** | **Приклад** | **Результат** |
| ***-match*** | Порівняння на збіг з урахуванням регулярний виразів у правому операнді | ***"книга" -match "ни"***  ***"ніс" -match "[к-н]іс"*** | ***$True***  ***$True*** |
| ***-notmatch*** | Порівняння на розбіжність з урахуванням регулярних виразів у правому операнді | ***"книга" -notmatch "лкн"*** | ***$False*** |
| ***-replace*** | Заміна або видалення символів в рядку - лівому операнді (ці оператори повертають змінений рядок). Якщо в якості правого операнда вказуються через кому два підрядка, то перший з них відповідає фрагменту, який потрібно змінити, а другий - рядку, який буде вставлений в результаті заміни. Якщо в якості правого операнда вказаний один підрядок, то він відповідає фрагменту рядка, який буде видалений | ***"рід" -replace "д", "к"***  ***"рід" –replace "рі"*** | ***"рік"***  ***"д"*** |

***Логічні оператори***

Іноді всередині однієї інструкції необхідно перевірити відразу кілька умов. Оператори порівняння можна з'єднувати один з одним за допомогою логічних операторів, наведених в табл. 9.5. При використанні логічного оператора **PowerShell** перевіряє кожну умову окремо, а потім обчислює значення інструкцій цілком, пов'язуючи умови за допомогою логічних операторів.

Табл. 9.5. 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)*** |

***Керуючі інструкції мови PowerShell***

У мові **PowerShell**, як і в будь-якій іншій алгоритмічній мові, є елементи, що дозволяють виконати логічне порівняння і зробити різні дії в залежності від його результату або дають можливість повторювати одну або кілька команд знову і знову.

***Інструкція If ... Elself ... Else***

Логічні порівняння лежать в основі практично всіх алгоритмічних мов програмування. У **PowerShell** за допомогою інструкції ***If*** можна виконувати певні блоки коду тільки в тому випадку, коли задана умова має значення ***$True*** (істина). Також можна задати одне або декілька додаткових умовних блоків. Відповідні їм умови будуть перевірятися, якщо всі попередні умови мали значення ***$False***. Нарешті, можна задати додатковий блок коду, який буде виконуватися в тому випадку, якщо жодна з умов не має значення ***$True***.

Синтаксис інструкції ***if*** в загальному випадку має такий вигляд:

***If (умова1)***

***{блок\_коду1}***

***[Elself (умова2)***

***{блок\_коду2}]***

***[Else***

***{ блок\_коду3}]***

При виконанні інструкції ***if*** перевіряється істинність умовного виразу ***умова1***. Якщо ***умова1*** має значення ***$True***, то виконується ***блок\_коду1***, після чого **PowerShell** завершує виконання інструкції ***if***. Якщо ***умова1*** має значення ***$False***, то **PowerShell** перевіряє істинність умовного виразу ***умова2***. Якщо ***умова2*** має значення ***$True***, то виконується ***блок\_коду2***, після чого **PowerShell** завершує виконання інструкції ***if***. Якщо і ***умова1***, і ***умова2*** мають значення ***$False***, то виконується ***блок\_коду3***, і виконання інструкції ***if*** завершується.

Наведемо приклад використання інструкції ***if*** в інтерактивному режимі роботи. Запишемо спочатку в змінну ***$а*** число ***10***:

***PS C:\> $a=10***

***PS C:\> if ($a -eq 15) {***

***>> 'Значення $a дорівнює 15'***

***>> } else***

***>> {'Значення $a не дорівнює 15'}***

***Значення $a не дорівнює 15***

З цього прикладу також видно, що в оболонці **PowerShell** в інтерактивному режимі можна виконувати інструкції, що складаються з декількох рядків (це може виявитися дуже корисним при налагодженні сценаріїв).

***Цикл While***

У **PowerShell** підтримуються кілька видів циклів. Найпростіший з них - цикл ***while***, в якому команди виконуються до тих пір, поки умова, що перевіряється, має значення ***$True***.

Інструкція ***while*** має наступний синтаксис:

***While (умова)***

***{блок\_команд}***

При виконанні інструкції ***while*** оболонка **PowerShell** обчислює розділ ***умова*** інструкції, перш ніж перейти до розділу ***блок\_команд***. Умова в інструкції приймає значення ***$True*** або ***$False***. До тих пір, поки умова має значення ***$True***, **PowerShell** повторює виконання розділу ***блок\_команд***.

Розділ ***блок\_команд*** інструкції ***While*** містить одну або кілька команд, які виконуються кожен раз при вході в цикл і його повторенні.

Наприклад, наступна інструкція ***while*** відображає числа від ***1*** до ***3***. Якщо змінна ***$val*** не була створена або була створена і ініціалізована значенням ***0***:

***PS C:\> while ($val -ne 3)***

***>> {***

***>> $val++***

***>> $val***

***>> }***

***1***

***2***

***3***

В даному прикладі ***умова*** (значення змінної ***$val*** не дорівнює ***3***) має значення ***$True***, поки ***$val*** дорівнює ***0***, ***1*** або ***2***. При кожному повторенні циклу значення ***$val*** збільшується на ***1*** з використанням унарного оператора збільшення значення ***++*** (***$val++***). При останньому виконанні циклу значення ***$val*** стає рівним ***3***. При цьому умова, що перевіряється, приймає значення ***$False***, і цикл завершується.

***Цикл Do ... While***

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***

Інструкція ***For*** в **PowerShell** реалізує ще один тип циклів – цикл з лічильником. Зазвичай цикл ***For*** застосовується для проходження по масиву і виконання певних дій з кожним із його елементів. У **PowerShell** інструкція ***For*** використовується не так часто, як в інших мовах програмування, так як колекції об'єктів зазвичай зручніше обробляти за допомогою інструкції ***ForEach***. Однак якщо необхідно знати, з яким саме елементом колекції або масиву ми працюємо на цій ітерації, то цикл ***For*** може допомогти.

Синтаксис інструкції ***For***:

***For (ініціалізація; умова; повторення) {блок\_команд}***

Класичний приклад:

***PS C:\> for ($i=0; $i -lt 5; $i++) { $i }***

***0***

***1***

***2***

***3***

***4***

***Цикл ForEach***

Інструкція ***ForEach*** дозволяє послідовно перебирати елементи колекцій. Найпростішим і найбільш часто використовуваним типом колекції, по якій проводиться переміщення, є масив. Зазвичай в циклі ***ForEach*** одна або кілька команд виконуються на кожному елементі масиву.

Особливістю циклу ***ForEach*** є те, що його синтаксис і робота залежать від того, де розташована інструкція ***ForEach***: поза конвеєром команд або всередині конвеєра.

***Інструкція ForEach поза конвеєром команд***

В цьому випадку синтаксис циклу ***ForEach*** має наступний вигляд:

***ForEach ($елемент in $колекція) {блок\_команд}***

В круглих дужках вказується колекція, по якій проводиться ітерація. При виконанні циклу ***ForEach*** система автоматично створює змінну ***$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.

Наприклад, цикл ***ForEach*** в наступному прикладі відображає значення в масиві з ім'ям ***$letterArray***:

***PS C:\> ForEach ($letter in $letterArray) {$letter}***

***a***

***b***

***c***

***d***

Інструкція ***ForEach*** може також використовуватися спільно командлетами, що повертають колекції елементів. Наприклад:

***PS С:\> $l=0; ForEach ($f in dir \*.txt) {$l+=$f.Length}***

***PS C:\> $l***

***2690555***

***Інструкція ForEach всередині конвеєра команд***

Якщо інструкція ***ForEach*** з'являється всередині конвеєра команд, то **PowerShell** використовує псевдонім ***ForEach***, відповідний командлету ***ForEach-Object***. Тобто в цьому випадку фактично виконується командлет ***ForEach-Object***, і вже не потрібно вказувати частина інструкції (***$елемент in $колекція***), так як елементи колекції блоку команд надає попередня команда в конвеєрі.

Синтаксис інструкції ***ForEach***, застосовуваної всередині конвеєра команд, в найпростішому випадку виглядає наступним чином:

***команда | ForEach {блок\_команд}***

Розглянутий вище приклад з підрахунком сумарного розміру текстових файлів з поточного каталогу для даного варіанту інструкції ***ForEach*** прийме наступний вигляд:

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

***PS С:\> $l***

***2690555***

У загальному випадку у псевдонімі ***ForEach*** може вказуватися не один блок команд, а три: початковий блок команд, середній блок команд і кінцевий блок команд. Початковий і кінцевий блоки команд виконуються один раз. А середній блок команд виконується кожен раз при черговій ітерації по колекції або масиву.

Синтаксис псевдоніма ***ForEach***, використовуваного в конвеєрі команд з початковим, середнім і кінцевим блоками команд виглядає наступним чином:

***команда | ForEach (початковий\_ блок\_команд} {середній\_блок\_команд) {кінцевий блок\_команд}***

Для цього варіанту інструкції ***ForEach*** наш приклад можна записати в такий спосіб:

***PS С:\> dir \*.txt | ForEach {$l=0} {$l+=length} {$l}***

***2690555***

***Мітки циклів, інструкції Break і Continue***

Інструкція ***Break*** дозволяє вийти з циклу будь-якого типу, не чекаючи закінчення його ітерацій. Розглянемо простий приклад:

***PS С:\> $i=0; While ($True) {If ($i++ -ge 3) { Break } $i }***

***1***

***2***

***3***

Умовою циклу ***While*** в даному випадку є логічна константа ***$True***, тому цей цикл не завершився б ніколи. Інструкція ***Break***, яка спрацьовує при досягненні змінної ***$i*** значення ***3,*** дозволяє вийти з даного циклу.

Інструкція ***Continue*** здійснює перехід до наступної ітерації циклу будь-якого типу. Наприклад:

***PS С:\> For ($i=0; $i -le 6; $i++) {If ($i -eq 3) {Continue} $i}***

***0***

***1***

***2***

***4***

***5***

***6***

У мові **PowerShell** підтримується можливість негайного виходу або переходу до наступної ітерації не тільки для одиночного циклу, але і для вкладених циклів. Для цього циклам присвоюються спеціальні мітки, які вказуються на початку рядка перед ключовим словом, що задає цикл того чи іншого типу. Такі мітки потім можна використовувати під вкладеними циклами спільно з інструкціями ***Break*** або ***Continue***, вказуючи, на який саме цикл повинні діяти дані інструкції. Розглянемо простий приклад:

***PS C:\> :outer While ($True) {***

***>> While ($True) {***

***>> Break outer***

***>> }***

***>> }***

***>>***

Тут інструкція ***Break*** здійснює вихід із зовнішнього циклу з міткою ***outer***. Якби мітка не було вказано, зовнішній цикл не завершився б ніколи.

***Інструкція Switch***

Інструкція ***switch***, яка об'єднує кілька перевірок умов всередині однієї конструкції, є в багатьох мовах програмування. Однак у мові **PowerShell** дана інструкція має потужні додаткові можливості:

* вона може використовуватися як аналог циклу, перевіряючи значення не єдиного елементу, а цілого масиву:
* вона може перевіряти елементи на відповідність шаблону з груповими символами або регулярними виразами:
* вона може обробляти текстові файли, використовуючи в якості елементів рядка з файлів, що перевіряються.

Розглянемо спочатку найпростішу форму інструкції ***Switch***, коли один скалярний вираз по черзі зіставляється з декількома умовами. Наприклад:

***PS C:\> $a=3***

***PS C:\> Switch ($a) {***

***>> 1 {"Один"}***

***>> 2 {"Два"}***

***>> 3 {"Три"}***

***>> 4 {"Чотири"}***

***>> }***

***>>***

***Три***

В даному прикладі значення змінної ***$а*** послідовно порівнюється з числами ***1***, ***2*** ,***3*** і ***4***. При збігу виконується відповідний блок коду, зазначений в фігурних дужках (в нашому випадку просто виводиться рядок).

Якщо для перевіряється значення справедливі кілька умов зі списку, то будуть виконані всі дії, зіставлені з цими умовами. Наприклад:

***PS C:\> $a=3***

***PS C:\> Switch ($a) {***

***>> 1 {"Один"}***

***>> 2 {"Два"}***

***>> 3 {"Три"}***

***>> 4 {"Чотири"}***

***>> 3 {"Ще раз три"}***

***>> }***

***>>***

***Три***

***Ще раз три***

Якщо потрібно обмежитися тільки першим збігом, то слід застосувати інструкцію ***Break***:

***PS C:\> $a=3***

***PS C:\> Switch ($a) {***

***>> 1 {"Один"}***

***>> 2 {"Два"}***

***>> 3 {"Три"; Break}***

***>> 4 {"Чотири"}***

***>> 3 {"Ще раз три"}***

***>> }***

***>>***

***Три***

В даному випадку перевірка умов всередині ***Switch*** переривається після знаходження першого відповідності. Якщо ж жодна відповідність ме знайдено, то інструкція ***Switch*** не виконує ніяких дій. За допомогою ключового слова ***Default*** можна задати дію за замовчуванням, яке буде виконуватися в тому випадку, коли, не знайдено жодна відповідність. Наприклад:

***PS С:\> Switch (3) {***

***>> 1 {"Один")***

***>> 2 {"Два"}***

***>> Default {"Ні один, ні два"}***

***>> }***

***>>***

***Ні один, ні два***

За замовчуванням в інструкції ***switch*** проводиться пряме порівняння з об'єктами, зазначеними в умови. Порівняння рядків при цьому проводиться без урахування регістрів символів, наприклад:

***PS C:\> Switch ('абв') {***

***>> 'абв' {"Перше співпадіння"}***

***>> 'АБВ' {"Друге співпадіння"}***

***>> }***

***>>***

***Перше співпадіння***

***Друге співпадіння***

Якщо при порівнянні слід врахувати регістр символів, то потрібно вказати параметр ***-CaseSensitive***:

***PS C:\> Switch -CaseSensitive ('абв') {***

***>> 'абв' {"Перше співпадіння"}***

***>> 'АБВ' {"Друге співпадіння"}***

***>> }***

***>>***

***Перше співпадіння***

Крім звичайного порівняння можна перевіряти елементи на відповідність шаблону з підстановочними символами. Для цього використовується перемикач ***-Wildcard***, наприклад:

***PS C:\> Switch -Wildcard ('абв') {***

***>> 'а\*' {"Починається з 'a'"}***

***>> '\*в' {"Закінується на 'в'"}***

***>> }***

***>>***

***Починається з 'a'***

***Закінується на 'в'***

Елемент (об'єкт), що перевіряється, доступний всередині інструкції ***Switch*** через спеціальну змінну ***$\_*** (нагадаємо, що в змінній з такою назвою зберігається і поточний елемент, який передається по конвеєру від одного командлеа до іншого). Наприклад:

***PS C:\> Switch -Wildcard ('абв') {***

***>> 'а\*' {"$\_ починається з 'a'"}***

***>> '\*в' {"$\_ закінується на 'в'"}***

***>> }***

***>>***

***абв починається з 'a'***

***абв закінується на 'в'***

Перемикач ***-Regex*** дозволяє перевіряти елементи на відповідність шаблону, що містить регулярні вирази. Попередній приклад можна записати і в такий спосіб:

***PS C:\> Switch -Regex ('абв') {***

***>> '^а' {"$\_ починається з 'а'"}***

***>> 'в$' {"$\_ закінчується на 'в'"}***

***>> }***

***>>***

***абв починається з 'а'***

***абв закінчується на 'в'***

Крім перевірок на простий збіг чи відповідність шаблону, інструкція ***Switch*** дозволяє виробляти більш складні перевірки, вказуючи замість шаблонів блоки коду на мові **PowerShell**. Значення, що перевіряється при цьому знову доступно через змінну ***$\_***. Розглянемо приклад:

***PS C:\> Switch (10) {***

***>> {$\_ -gt 5} {"$\_ більше 5"}***

***>> {$\_ -lt 20} {"$\_ менше 20"}***

***>> 10 {"$\_ дорівнює 10"}***

***>> }***

***>>***

***10 більше 5***

***10 менше 20***

***10 дорівнює 10***

До теперішнього моменту всі значення, які ми перевіряли в інструкції ***Switch***, були скалярними величинами. Мова **PowerShell** допускає використання в якості значення, що перевіряється, масиву елементів, причому масиви можуть задаватися явно або виходити в результаті виконання будь-якої команди, скажімо, шляхом зчитування рядків з текстового файлу. Розглянемо приклад:

***PS C:\> $arr=1,2,3,4,5,6***

***PS C:\> Switch ($arr) {***

***>> {$\_%3} {"$\_ не ділиться на 3"}***

***>> Default {"$\_ ділиться на 3"}***

***>> }***

***>>***

***1 не ділиться на 3***

***2 не ділиться на 3***

***3 ділиться на 3***

***4 не ділиться на 3***

***5 не ділиться на 3***

***6 ділиться на 3***

В даному випадку масив цілих чисел задається явним перерахуванням своїх елементів. Всі елементи масиву по черзі перевіряються всередині інструкціі ***Switch***: якщо залишок від ділення поточного елемента на ***3*** не дорівнює нулю, то виводиться повідомлення ***"$\_ не ділиться на 3"***. Де замість ***$\_*** підставляється, як зазвичай, значення елемента, що перевіряється. В іншому випадку видається повідомлення ***"$ \_ ділиться на 3"***.

Наведемо ще один приклад, коли масив перевіряються елементів є результатом виконання команди **PowerShell**. Нехай нам потрібно дізнатися кількість файлів з розширеннями ***txt*** і ***log***, що знаходяться в системному каталозі Windows. Спочатку Обнуляємо змінні-лічильники:

***PS С:\> $txt=$log=0***

Виконаємо тепер відповідну інструкцію ***switch***. Колекцію файлів з системного каталогу **Windows** отримаємо за допомогою команди ***dir $env:SystemRoot*** (нагадаємо, що у змінній середовища ***SystemRoot*** зберігається шлях до потрібного каталогу). Файли з розширеннями ***txt*** і ***log*** будемо відбирати, перевіряючи на відповідність шаблонів з підстановочними символами (***\*.txt*** і ***\*.log***), і у відповідних блоках коду будемо збільшувати значення змінних ***$txt*** і ***$log***:

***PS С:\> Switch -Wildcard (dir $env:SystemRoot) {***

***>> \*.txt ($txt++}***

***>> \*.log {$log++}***

***>> }***

***>>***

Виведемо тепер значення змінних ***$txt*** і ***$log***:

***PS С:\> "txt-файли: $txt log-файли: $log"***

***txt-файли: 21 log-файли: 156***

***Функції у PowerShell***

До теперішнього моменту ми працювали переважно зі стандартними скомпільованими Командлети. функціональність яких ми не можемо змінити, гак як їх вихідний код в оболонці **PowerShell** недоступний. Функції та сценарії - це два інших типи команд **PowerShell**, які можна створювати і змінювати на свій розсуд, керуючись мовою **PowerShell**.

Слід відразу звернути увагу, що функції в **PowerShell** мають деякі особливості в порівнянні з функціями в традиційних мовах програмування, оскільки **PowerShell** - це в першу чергу оболонка. У традиційних мовах функція, як правило, є аналогом методу об'єкта, а в **PowerShell** функція - це команда. Звідси відмінність в способах виконання функцій і передачі їм аргументів.

Зазвичай функції в традиційних мовах повертають єдине значення того чи іншого типу. "Значенням" функції **PowerShell** може бути цілий масив різних об'єктів, так як кожний вираз, що обчислюється у функції, поміщує свій результат у вихідний потік.

Крім того, функції в **PowerShell** діляться на кілька типів відповідно до їх поведінки всередині конвеєра команд.

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 (даний список повинен бути заключений у фігурні дужки). Наприклад, створимо функцію ***MyFunс***:

***PS C:\> Function MyFunc {"Всім привіт!"}***

Для виклику цієї функції достатньо просто ввести її ім’я:

***PS C:\> MyFunc***

***Всім привіт!***

Відзначимо, що в багатьох мовах програмування при виконанні функції після її імені потрібно вказувати круглі дужки. У **PowerShell** цього робити не можна:

***PS C:\> MyFunc()***

***An expression was expected after '('.***

***At line:1 char:8***

Функції можуть працювати з аргументами, які передаються їм при запуску, причому підтримуються два варіанти обробки таких аргументів: за допомогою змінної ***$Args*** і шляхом завдання формальних параметрів.

***Обробка аргументів функцій за допомогою змінної $Args***

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

***PS C:\> Function MyFunc {"Привіт, $Args!"}***

Так як змінна ***$Args*** поміщена в рядок в подвійних лапках, то при 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!***

Як бачите, три зазначених нами параметра (три елементи масиву ***$Args***) поміщені у вихідний рядок і розділені між собою проміжками. Можна змінити символ, що розділяє елементи при їх підстановці, перевизначивши значення спеціальної змінної ***$OFS***:

***PS C:\> Function MyFunc {***

***>> $OFS=","***

***>> "Привіт, $Args!"}***

***>>***

***PS C:\> MyFunc Андрій Сергій Іван***

***Привіт, Андрій,Сергій,Іван!***

Зверніть увагу, що на відміну від традиційних мов програмування. функції в **PowerShell** є командами (це не методи об'єктів!), тому їх аргументи вказуються через проміжок без додаткових круглих дужок і виділення символьних рядків лапками. Щоб наочно переконатися в цьому, запустимо функцію ***MyFunc*** наступним чином:

***PS C:\> MyFunc "Андрій","Сергій","Іван"***

***Привіт, System.Object[]!***

Нагадаємо, що масиви в **PowerShell** задаються перерахуванням своїх елементів, а круглі дужки, що оточують який-небудь вираз, означають, що цей вираз має бути обчислено. Тому помилки при такому виконанні функції не виникає, однак результат її виконання виявляється зовсім іншим, адже в даному випадку в функцію передаються не три символьних аргументи, а один аргумент, який є масивом з трьох елементів!

Так як змінна ***$Args*** є масивом, то всередині функції можна звертатись до окремих аргументів по їх порядковому номеру (нагадаємо, що нумерація елементів масивів починається з нуля), а за допомогою методу Count визначати загальну кількість аргументів, переданих функції. Для прикладу створимо функцію SumArgs, яка буде повідомляти про кількість своїх аргументів і обчислювати їх суму:

***PS С:\> Function SumArgs {Кількість аргументів: $($Args.Count)"***

***>> $n=0***

***>> For($i=0; $i -lt $Args.Count; $i++) { $n+=$Args[$i] }***

***>>"Сума аргументів: $n"}***

***>>***

Запустимо функцію ***SumArgs*** з трьома числовими аргументами:

***PS C:\> SumArgs 1 2 3***

***Кількість аргументів: 3***

***Сума аргументів: 6***

Крім використання масиву ***$Args*** в ***PowerShell*** підтримується альтернативний підхід до обробки аргументів функцій - за допомогою завдання формальних параметрів.

***Формальні параметри функцій***

У **PowerShell**, як і в більшості інших мов програмування, при описі 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}***

При виконанні функції ***Subtract*** її формальні параметри будуть замінені фактичними аргументами, які визначаються або за позиції в командному рядку, або по імені.

Наприклад:

***PS C:\> Substract 10 2***

***8***

В цьому випадку відповідність формальних параметрів фактично переданим аргументів визначається по позиції: замість першого параметра ***$From*** підставляється число ***10***, замість другого параметра ***$Count*** підставляється число ***2***.

При вказівці аргументів можна використовувати імена формальних параметрів (порядок вказівки аргументів при цьому стає несуттєвим), наприклад:

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

***8***

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

***2***

Якщо двічі вказати ім'я одного і того ж параметра, то виникне помилка:

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

***Substract : Cannot bind parameter because parameter 'From' is specified more than once. To provide multiple values to parameters that can accept multiple values, use the array syntax. For example, "-parameter value1,value2,value3".***

***At line:1 char:25***

При виконанні функції можливий і третій варіант завдання аргументів, коли для деяких задаються імена, а деякі визначаються по позиції в командному рядку. При цьому діє наступний алгоритм:

* Все іменовані аргументи зіставляються відповідним формальним параметрам і видаляються зі списку аргументів;
* Решта параметрів (безіменні або ті, що мають ім'я, якому не відповідає жоден формальний параметр) зіставляються формальним параметрам по позиції.

Наприклад:

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

***8***

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

***8***

За замовчуванням функції **PowerShell**, як і інші команди, поводяться поліморфним чином. Наприклад, визначимо функцію ***Add***, що складає два своїх аргументи:

***PS C:\> Function Add ($x, $y) {$x+$y}***

Виконаємо цю функцію з аргументами-числами і аргументами-рядками:

***PS C:\> Add 2 3***

***5***

***PS C:\> Add "2" "3"***

***23***

У першому випадку функція ***Add*** повертає число ***5***, а в другому - рядок ***"23"***.

При необхідності при оголошенні функції можна явно задати тип формальних параметрів. Наприклад, визначимо функцію ***IntAdd***, яка буде складати два свої цілочисельні аргументи:

***PS C:\> Function IntAdd([int] $x, [int] $y) {$x+$y}***

Як і в попередньому випадку, викличемо цю функцію з аргументамі-числами і з аргументами-рядками:

***PS C:\> IntAdd 2 3***

***5***

***PS C:\> IntAdd "2" "3"***

***5***

Як бачите, результат виходить один і той же, так як символьні аргументи перетворюються до цілочисельного типу. Якщо перетворення виконати не вдається, то виникне помилка.

При виконанні функції може бути вказано більшу кількість фактичних аргументів, ніж було задано формальних параметрів. При цьому "зайві" аргументи будуть поміщені в масив ***$Args***. Для ілюстрації розглянемо функцію ***ShowArgs*** з двома формальними параметрами:

***PS C:\> Function ShowArgs($x,$y) {***

***>> "Перший аргумент: $x"***

***>> "Другий аргумент: $y"***

***>> "Інші аргументи: $Args" }***

***>>***

Викличемо функцію з чотирма аргументами:

***PS C:\> ShowArgs 1 2 3 4***

***Перший аргумент: 1***

***Другий аргумент: 2***

***Інші аргументи: 3 4***

Додаткові аргументи дійсно зберігаються в масиві ***$Args***.

При оголошенні формальних параметрів можна вказати значення, які будуть приймати ці параметри за замовчуванням (якщо явно не вказано відповідний фактичний аргумент). Наприклад, перевизначити функцію ***Add*** як функцію з двома формальними параметрами ***$х*** і ***$у***, які за замовчанням ініціалізуються числами ***2*** і ***3***:

***PS C:\> Function Add($x=2,$y=3){$x+$y}***

Якщо почати його використання без аргументів, то обидва параметри ініціалізуються значеннями за замовчуванням і функція повертає число ***5***:

***PS C:\> Add***

***5***

Якщо запустити функцію ***Add*** з одним аргументом, то вказане значення буде присвоєне параметру ***$х***, а параметр ***$у*** знову инициализируется значенням за замовчуванням (***$у=3***):

***PS C:\> Add 5***

***8***

При вказівці двох параметрів функція ***Add*** поверне їх суму:

***PS C:\> Add 6 6***

***12***

У функціях також можна використовувати параметри-перемикачі, які повинні мати тип ***SwitchParameter***, що має псевдонім ***[Switch]***. Значеннями перемикача можуть бути тільки ***$True*** або ***$False***, тому форматувати подібний формальний параметр пе потрібно.

Для прикладу визначимо функцію ***MyFunc*** з одним параметром-перемикачем, що визначає поведінку функції:

***PS C:\> Function MyFunk([Switch] $Recurse) {***

***>> if ($Recurse) {***

***>> "Рекурсивний варіант функції"***

***>> }***

***>> else***

***>> {***

***>> "Звичайний варіант функції"***

***>> }***

***>> }***

Запустивши функцію MyFunc без параметра, ми отримаємо повідомлення, що вона працює в звичайному режимі:

***PS C:\> MyFunk***

***Звичайний варіант функції***

Якщо вказано параметр ***-Recurse***, то функція ***MyFunc*** буде працювати в альтернативному режимі:

***PS C:\> MyFunk -Recurse***

***Рекурсивний варіант функції***

***Значення, що повертаються***

У традиційних мовах програмування функція зазвичай повертає єдине значення певного типу. В оболонці **PowerShell** справа йде інакше – тут результати всіх виразів або конвеєрів, що обчислюються всередині функції, направляються в вихідний потік. Розглянемо, наприклад, функцію ***MyFunc***, в якій обчислюються три числові вирази:

***PS C:\> Function MyFunk{1+2;3\*3;12/4}***

Як бачите, в цій функції явно не повертається жодне значення. Запустимо функцію ***MyFunc***:

***PS C:\> MyFunk***

***3***

***9***

***3***

На екран виводяться три рядки з результатами обчислень. Тепер запустимо цю функцію і збережемо результат її роботи в змінній ***$Result***:

***PS C:\> $Result=MyFunk***

Перевіримо тип змінної ***$Result***:

***PS C:\> $Result.GetType().FullName***

***System.Object[]***

Для того, щоб певне значення не потрапляло у вихідний потік, слід скористуватись командлетом ***Write-Host***, який не додає рядки у вихідний потік.

***PS С:\> Function MyFunc {***

***>> $name=Read-Host "Введіть своє ім’я"***

***>> Write-Host "Доброго дня, $name!"***

***>> $name***

***>> }***

***PS C:\> MyFunc***

***Введіть своє ім’я: Гліб***

***Доброго дня, Гліб!***

***Гліб***

***PS C:\> $Result=MyFunc***

***Введіть своє ім’я: Гліб***

***Доброго дня, Гліб!***

***PS C:\> $Result***

***Гліб***

Як бачите, командлет ***Write-Host*** виводить рядок безпосередньо на консоль, а не в вихідний потік, тому в даному випадку функція ***MyFunc*** повертає єдиний рядок - значення змінної ***$name***.

У мові **PowerShell** є інструкція ***Return***, що виконує негайний вихід з функції.

***Функції всередині конвеєра команд***

Вся ідеологія PowerShell побудована на застосуванні конвеєрів команд, і функції тут не є винятком: їх теж можна використовувати всередині конвеєрів. При цьому для прийому всередині функції вхідного потоку об'єктів, переданих від іншої команди, служить змінна ***$Input***. У цієї змінної буде міститися колекція входять об'єктів. Розглянемо приклад:

***PS C:\> Function Sum {***

***>> $n=0***

***>> ForEach ($i in $Input) {$n+=$i}***

***>> $n***

***>> }***

***PS C:\> 1..10 | Sum***

***55***

***PS C:\> Function Sum2 {***

***>> $n=0***

***>> While ($Input.MoveNext()) {***

***>> $n+=$Input.Current***

***>> }***

***>> $n***

***>> }***

Запустимо функцію ***Sum2*** і переконаємося, що вона видає той же результат, що і функція ***Sum***:

***PS C:\> 1..10 | Sum2***

***55***

Таким чином, написати функцію для роботи всередині конвеєра неважко (достатьо знати про призначення змінної ***$Input***). Однак є один нюанс. При отриманні даних від попередньої команди функція призупиняє конвеєр і запускається тільки один раз, коли сформована вся колекція вхідних елементів. Іншими словами, звичайна функція не підтримує в повній мірі механізм конвейеризации, коли елемент обробляється в конвеєрі, не чекаючи створення наступних елементів. Для більш ефективної роботи всередині конвеєра функцію слід оформляти у вигляді фільтра.

***Фільтри в PowerShell***

**Фільтр** - це функція особливого виду, яка, перебуваючи всередині конвеєра, запускається для кожного, хто входить елемента (звичайні функції запускаються один раз для всієї сукупності елементів, сформованої попередньою командою конвеєра).

Синтаксично фільтри відрізняються від функцій лише тим. що замість ключового слова ***Function*** вказується слово ***Filter***:

***Filter ім'я\_фільтра (параметри) {блок\_кода}***

Однак алгоритми роботи звичайної функції і фільтра розрізняються, якщо вони знаходяться всередині конвеєра. У звичайній функції доступ до вхідних елементів конвеєра здійснюється через колекцію ***$Input***. У фільтрі ж визначена змінна ***$\_***, що відповідає поточному елементу конвеєра, що проходить через даний фільтр.

Розглянемо приклад. Напишемо фільтр, який буде подвоювати числа, що проходять через нього по конвеєру. Такий фільтр буде складатися всього з одного виразу:

***PS C:\> Filter Double {$\_\*2}***

Перевіримо роботу даного фільтра:

***PS C:\> 1..4 | Double***

***2***

***4***

***6***

***8***

***PS C:\> 1..4 | Double | Double***

***4***

***8***

***12***

***16***

Функціональність фільтрів дуже схожа на функціональність командлету ***ForEach-Object***, який можна вважати безіменним фільтром.

***Сценарії PowerShell***

Сценарії **PowerShell** є текстовими файли з розширенням ***ps1***, в 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, переносячи потім готовий код у зовнішній текстовий файл. Такий підхід значно спрощує вивчення мови і налагодження сценаріїв, дозволяючи відразу бачити результат виконання окремих частин сценарію.

Створити файл зі сценарієм **PowerShell** можна різними способами:

* скористатися зовнішнім текстовим редактором (наприклад, стандартним Блокнотом Windows), вручну ввести потрібні команди і зберегти файл з розширенням ***ps1***;
* виконати потрібні команди в оболонці **PowerShell**, скопіювати їх з консолі в буфер Windows і вставити в текстовий файл, відкритий в зовнішньому текстовому редакторі (прийоми роботи з оболонки з буфером Windows обговорювалися в розділі 2). і потім зберегти отриманий в результаті файл з розширенням ***ps1***;
* працюючи в оболонці **PowerShell**, включити за допомогою командлету ***Start- Transcript*** режим протоколювання команд. У результаті буде створено зовнішній файл з усіма командами, що запускаються в сеансі роботи. З цього файлу можна скопіювати потрібні команди в інший текстовий файл і зберегти його з розширенням ***ps1***;
* перебуваючи в оболонці **PowerShell**, оформити команди **PowerShell** у вигляді рядків (звичайних або автономних) і перенаправити за допомогою символів ***>*** і ***>>*** ці рядки в зовнішній файл з розширенням ***ps1***.

Скористаємося останнім із запропонованих варіантів і створимо в каталозі ***C:\Script*** найпростіший сценарій ***test.ps1***. який буде складатися з єдиного рядка:

***"Цей рядок друкується зі сценарію PowerShell"***

Тепер запишемо потрібний рядок (включаючи подвійні лапки) в файл ***test.ps1***:

***PS C:\Script> ‘"Цей рядок друкується зі сценарію PowerShell"’ > test.ps1***

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

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

***PS C:\Script> Get-ExecutionPolicy***

***Restricted***

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

***PS C:\Script> Set-ExecutionPolicy RemoteSigned***

Даний 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***

При запуску сценаріїв **PowerShell** шлях до файлу з кодом потрібно завжди вказувати явно, навіть якщо сценарій знаходиться в поточному каталозі, так як це запобігає можливому несанкціонованому запуску іншої виконуваної програми з аналогічним ім'ям, що знаходиться, наприклад, в системному каталозі. Тому запустимо сценарій наступним чином (точка відповідає поточному каталогу):

***PS C:\Script> .\test.psl***

***Цей рядок друкується зі сценарію PowerShell***

При цьому можна навіть не вказувати розширення ***.ps1***. А можна вказати і повний шлях до файлу.

Сценарії **PowerShell** можна запускати безпосередньо з командного рядка інтерпретатора **cmd.exe** або за допомогою пункту **Виконати** меню **Пуск**. Для цього потрібно вказати повний шлях до цього сценарію в якості параметра програми **powershell.exe** (повний шлях до **powershell.exe** можна не вказувати).

Нагадаємо, що командні файли інтерпретатора **cmd.exe**, так само як і сценарії **WSH** на мовах **VBScript** або **JScript**, можна було запускати з Провідника Windows, просто клацаючи мишею на значках цих сценаріїв. Зі сценаріями **PowerShell** цей метод не працює - якщо двічі клацнути мишею на значку сценарію **PowerShell**, то він не запуститься, а відкриється для редагування в Блокноті (з точки зору безпеки це правильний підхід, що дозволяє запобігти випадковий запуск сценарію).

***Передача аргументів в сценарії***

Розбір і обробка аргументів, переданих в сценарії, проводиться практично так само, як і в 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 проміжками. Змінна ***$Args*** всередині сценарію містить масив, елементами якого є аргументи функції, зазначені при 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 ж 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, так як тут все вміст файлу є тілом сценарію, тому формальні параметри задаються за допомогою спеціальної інструкції ***Param***. Ця інструкція повинна бути найпершою командою в 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***

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

***Вихід із сценаріїв***

У звичайному режимі вихід зі сценарію, як і з функції, відбувається після виконання останньої інструкції в ньому. Нагадаємо, що у функції можна було скористатися інструкцією ***Return*** для примусового завершення роботи. Аналог цієї інструкції для сценаріїв - інструкція ***Exit***, що дозволяє завершити роботу сценарію і. при необхідності, повернути певний код повернення. Код повернення може аналізуватися в зовнішніх програмах (наприклад, командних файлах або сценаріях **WSH**), що запускають сценарій **PowerShell**.

***Коментарі***

У мові **PowerShell** коментарі починаються зі знака ***#***. Всі символи, що стоять після цього знака, розглядаються як частина коментаря і не інтерпретуються системою. Наприклад:

***PS C:\> # Весь рядок - коментар. Вираз 2+2 не обчислюється***

***PS C:\> 2+2 # Коментар в кінці рядка. Вираз перед ним обчислюється***

***4***

***Доступ з PowerShell до зовнішніх об'єктів***

Однією з основних завдань, для вирішення яких створювалася оболонка **PowerShell**, було отримання з командного рядка доступу до різних об'єктних структур, підтримуваним операційною системою **Windows**. Раніше до подібних об'єктів можна було звертатися або з повноцінних додатків за допомогою інтерфейсу прикладного програмування (API), або зі сценаріїв **WSH**. У будь-якому випадку для використання зовнішніх об'єктів доводилося писати програмний код і вивчати їх структуру, що значно ускладнювало роботу з ними і перешкоджало широкому поширенню технологій автоматизації серед системних адміністраторів і користувачів Windows.

Для вирішення цієї проблеми в **PowerShell** були розроблені спеціальні командлети, що дозволяють в інтерактивному режимі з оболонки звертатися до об'єктів WMI, СОМ і .NET.

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

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

Виконуючи командлет ***New-Object***, інтерпретатор **PowerShell** отримує з системного реєстру шлях до файлів потрібної бібліотеки типів. Потім за допомогою цієї бібліотеки в пам'ять завантажується екземпляр запитуваного об'єкта, і його інтерфейси стають доступними для використання в **PowerShell**. Посилання на створений об'єкт зберігається в змінній: надалі, використовуючи цю змінну, ми отримуємо доступ до властивостей і методів об'єкта, а також до його вкладених об'єктів (якщо вони є).

Подивимося, які властивості і методи є у СОМ-об'єкту ***WScript.Shell***. Для цього скористаємося, як зазвичай, командлетом ***Get-Member***, передавши йому по конвеєру змінну ***$shell***, в якій збережене посилання на даний СОМ-об'єкт:

***PS C:\> $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)***

***ExpandEnvironmentStrings Method string ExpandEnvironmentStrings (string)***

***LogEvent Method bool LogEvent (Variant, string, string)***

***Popup Method int Popup (string, Variant, Variant, Variant)***

***RegDelete Method void RegDelete (string)***

***RegRead Method Variant RegRead (string)***

***RegWrite Method void RegWrite (string, Variant, Variant)***

***Run Method int Run (string, Variant, Variant)***

***SendKeys Method void SendKeys (string, Variant)***

***Environment ParameterizedProperty IWshEnvironment Environment (Variant) {get}***

***CurrentDirectory Property string CurrentDirectory () {get} {set}***

***SpecialFolders Property IWshCollection SpecialFolders () {get}***

Спробуємо скористатися яким-небудь методом COM-об'єкта ***WScript.Shell***. Наприклад, метод ***CreateShortcut*** дозволяє швидко створювати ярлики для папок і файлів. Для прикладу ми створимо на робочому столі активного користувача ярлик ***PSHome.lnk*** для папки, в якій встановлена оболонка **PowerShell**.

Шлях до робочого столу можна визначити різними способами, наприклад, за допомогою завжди визначеної в **PowerShell** змінної ***$Ноme***, в якій зберігається шлях до папки активного користувача:

***PS C:\> $lnk = $Shell.CreateShortcut("$Ноme\Desktop\РSНоme.lnk")***

***Використання об'єктів .NET***

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. Найбільш часто використовуються збірки завантажуються в **PowerShell** автоматично, їх список можна побачити за допомогою статичного методу ***getAssemblies()*** наступним чином:

***PS С:\> [AppDomain]::CurrentDomain.getAssemblies()***

Наприклад, для звернення до об'єктів ***WinForms*** потрібно за допомогою методу ***LoadWithPartialName*** завантажити збірку, що підтримують ці 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")***

Будь-яке графічне додаток повинен мати головну форму - об'єкт типу ***Windows.Forms.Form***. Ми збережемо цю форму в змінній ***$form***:

***PS С:\> $form = New-Object Windows.Forms.Form***

У заголовок форми (властивість ***Text*** об'єкта ***$form***) запишемо рядок ***"Перша форма"***:

***PS C:\> $form.Text = "Перша форма"***

Тепер створимо кнопку (об'єкт типу ***Windows.Forms.Button***) з написом ***"Натисни!***":

***PS C:\> $button = New-Object Windows.Forms.Button***

***PS C:\> $button.Text = "Натисни!"***

Визначимо тепер дію, яка виконуватиметься при натисканні кнопки. Для цього потрібно написати обробник події ***Click*** кнопки (тобто вказати, які команди повинні виконатися при натисканні на кнопку). Оброблювач подій - це спеціальний метод з назвою ***Аdd\_подія***. Нехай в нашому випадку натискання кнопки буде приводити до закриття форми (метод ***Close*** об'єкта ***$form***):

***PS C:\> $button.Add\_Click({$form.Close()})***

Тепер помістимо кнопку на форму за допомогою методу ***Add*** колекції ***Controls*** об'єкта ***$form***:

***PS C:\> $form.Controls.Add($button)***

Для того щоб наша форма стала активною при відображенні, потрібно визначити обробник події ***Shown***, помістивши в цей обробник виклик методу ***Activate*** об'єкта ***$form***:

***PS C:\> $form.Add\_Shown({$form.Activate()})***

Тепер залишилося викликати метод ***ShowDialog*** для виведення нашої форми на екран:

***PS C:\> $form.ShowDialog()***

