Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический
университет имени П. О. Сухого»

Кафедра «Информационные системы»

К. С. Курочка, Д. А. Литвинов

ОПЕРАЦИОННЫЕ СИСТЕМЫ

ПОСОБИЕ
по одноименному курсу
для студентов специальности 1-40 01 02
«Информационные системы и технологии
(по направлениям)»

Электронный аналог печатного издания

Гомель 2009

УДК 004.451(075.8)
ББК 32.973-018.2я73
К93

Рекомендовано к изданию научно-методическим советом факультета автоматизированных и информационных систем ГГТУ им. П. О. Сухого
(*протокол № 10 от 29.06.2009 г.*)

Рецензенты: зав. каф. «Промышленная электроника» ГГТУ им. П. О. Сухого
канд. техн. наук *Ю. В. Крышнев*

Курочка, К. С.

К93 Операционные системы : пособие по одноим. курсу для студентов специальности 1-40 01 02 «Информационные системы и технологии (по направлениям)» / К. С. Курочка, Д. А. Литвинов. – Гомель : ГГТУ им. П. О. Сухого, 2009. – 66 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: http://lib.gstu.local. – Загл. с титул. экрана.

ISBN 978-985-420-874-9.

Приведена история развития и классификация операционных систем, краткие теоретические и практические сведения, необходимые для освоения командного режима работы в операционных системах семейства Windows и Linux/Unix.

Для студентов специальности 1-40 01 02 «Информационные системы и технологии (по направлениям)».

УДК 004.451(075.8)
ББК 32.973-018.2я73

ISBN 978-985-420-874-9

ВВЕДЕНИЕ

В современных условиях интенсивного развития вычислительной техники нельзя представить квалифицированного специалиста в области информационных технологий без знаний и умения использовать средства и инструментарий, предоставляемые операционными системами. Одними из таких базовых средств являются команды операционной системы, с помощью которых можно выполнять множество функций. Зачастую, использование командного процессора является наиболее простым и мощным средством выполнения ряда действий по настройке операционной системы, управлению файлами, правилами безопасности и конфигурированию сетевого аппаратного обеспечения.

Пособие содержит краткие теоретические и практические сведения, необходимые для освоения командного режима работы в операционных системах семейства Windows и Linux/Unix.

1. ИСТОРИЯ РАЗВИТИЯ ОПЕРАЦИОННЫХ СИСТЕМ

Операционная система (ОС) – это программное обеспечение, реализующее рациональное использование аппаратных средств вычислительной системы прикладными программами пользователя.

Основная функция ОС – это оптимальное распределение ресурсов вычислительной системы между пользовательскими задачами.

В операционных системах можно выделить следующие блоки:

- планирование процессов и потоков;
- управление внутренней памятью (менеджер памяти);
- управление внешней памятью (поддержка файловой системы);
- управление вводом/выводом;
- управление графической видеосистемой;
- управление сетевыми ресурсами;
- управление другими ресурсами (принтеры, сканеры и т. п.).

Первый период (1945–1955)

В середине 40-х годов были созданы первые ламповые вычислительные устройства. В то время одна и та же группа людей участвовала и в проектировании, и в эксплуатации, и в программировании вычислительной машины. Программирование осуществлялось исключительно на машинном языке. Об операционных системах не было и речи, все задачи организации вычислительного процесса решались вручную каждым программистом с пульта управления. Не было никакого другого системного программного обеспечения кроме библиотек математических и служебных подпрограмм.

Второй период (1955–1965)

С середины 50-х годов при создании компьютеров стали использоваться полупроводниковые элементы. Компьютеры второго поколения стали более надежными, теперь они смогли непрерывно работать настолько долго, чтобы на них можно было возложить выполнение действительно практически важных задач. Именно в этот период произошло разделение персонала на программистов и операторов, эксплуатационщиков и разработчиков вычислительных машин. В эти годы появились первые алгоритмические языки, а, следовательно, и первые системные программы – компиляторы. Стоимость процессорного времени возросла, что потребовало уменьшения непроизводительных затрат времени между запусками программ. Появились первые системы пакетной обработки, которые просто автоматизировали запуск одной программы за другой и тем самым увеличивали ко-

эффициент загрузки процессора. Системы пакетной обработки явились прообразом современных операционных систем, они стали первыми системными программами, предназначенными для управления вычислительным процессом. В ходе реализации систем пакетной обработки был разработан формализованный язык управления заданиями, с помощью которого программист сообщал системе и оператору, какую работу он хочет выполнить на вычислительной машине. Совокупность нескольких заданий, как правило, в виде колоды перфокарт, получила название пакета заданий.

Третий период (1965–1980)

В середине 60-х годов в технической базе произошел переход от отдельных полупроводниковых элементов типа транзисторов к интегральным микросхемам, что дало гораздо большие возможности новому, третьему поколению компьютеров, как в плане повышения быстродействия, так и в плане надежности, уменьшения размеров самого компьютера и снижения его цены.

Для этого периода характерно также создание семейств программно-совместимых машин. Первым семейством программно-совместимых машин, построенных на интегральных микросхемах, явилась серия машин IBM/360. Созданное в начале 60-х годов это семейство значительно превосходило машины второго поколения по критерию цена/производительность. Вскоре идея программно-совместимых машин стала общепризнанной. Программная совместимость требовала и совместимости операционных систем.

Важнейшим достижением ОС данного поколения явилась реализация мультипрограммирования. Мультипрограммирование – это способ организации вычислительного процесса, при котором на одном процессоре попеременно выполняются несколько программ.

Наряду с мультипрограммной реализацией систем пакетной обработки появился новый тип ОС – системы разделения времени. Вариант мультипрограммирования, применяемый в системах разделения времени, нацелен на создание для каждого отдельного пользователя иллюзии единоличного использования вычислительной машины.

Четвертый период (1980–настоящее время)

Следующий период в эволюции операционных систем связан с появлением больших интегральных схем (БИС). В эти годы произошло резкое возрастание степени интеграции и удешевление микросхем. Компьютер стал доступен отдельному человеку, и наступила эра персональных компьютеров. Компьютеры стали широко использоваться

неспециалистами, что потребовало разработки «дружественного» программного обеспечения.

В середине 80-х стали бурно развиваться сети персональных компьютеров, работающие под управлением сетевых или распределенных ОС.

2. КЛАССИФИКАЦИЯ ОПЕРАЦИОННЫХ СИСТЕМ

В зависимости от особенностей использованного алгоритма управления процессором операционные системы делят на многозадачные и однозадачные, многопользовательские и однопользовательские, на системы, поддерживающие многопоточную обработку и не поддерживающие ее, на многопроцессорные и однопроцессорные системы.

На свойства операционной системы непосредственное влияние оказывают аппаратные средства, на которые она ориентирована. По типу аппаратуры различают операционные системы персональных компьютеров, мини-компьютеров, мейнфреймов, кластеров и сетей ЭВМ. Среди перечисленных типов компьютеров могут встречаться как однопроцессорные варианты, так и многопроцессорные. В любом случае специфика аппаратных средств, как правило, отражается на специфике операционных систем.

Многозадачные ОС подразделяются на три типа в соответствии с использованными при их разработке критериями эффективности:

- системы пакетной обработки (например, ОС IBM /360);
- системы разделения времени (UNIX, VMS);
- системы реального времени (QNX, RT/11).

Системы пакетной обработки предназначались для решения задач в основном вычислительного характера, не требующих быстрого получения результатов. Главной целью и критерием эффективности систем пакетной обработки является максимальная пропускная способность, т. е. решение максимального числа задач в единицу времени.

Системы разделения времени призваны исправить основной недостаток систем пакетной обработки – изоляцию пользователя-программиста от процесса выполнения его задач. Каждому пользователю системы разделения времени предоставляется терминал, с которого он может вести диалог со своей программой. Так как в системах разделения времени каждой задаче выделяется только квант процессорного времени, ни одна задача не занимает процессор надолго, и время ответа оказывается приемлемым.

Системы реального времени применяются для управления различными техническими объектами. Критерием эффективности для систем реального времени является их способность выдерживать заранее заданные интервалы времени между запуском программы и получением результата (управляющего воздействия). Это время называется временем реакции системы, а соответствующее свойство системы – реактивностью. Для этих систем мультипрограммная смесь представляет собой фиксированный набор заранее разработанных программ, а выбор программы на выполнение осуществляется, исходя из текущего состояния объекта или в соответствии с расписанием плановых работ.

ОС различаются по способам построения ядра системы – монолитное ядро или микроядерный подход. Большинство ОС использует монолитное ядро, которое компонуется как одна программа, работающая в привилегированном режиме и использующая быстрые переходы с одной процедуры на другую, не требующие переключения из привилегированного режима в пользовательский и наоборот. Альтернативой является построение ОС на базе микроядра, работающего также в привилегированном режиме и выполняющего только минимум функций по управлению аппаратурой, в то время как функции ОС более высокого уровня выполняют специализированные компоненты ОС – серверы, работающие в пользовательском режиме. При таком построении ОС работает более медленно, так как часто выполняются переходы между привилегированным режимом и пользовательским, зато система получается более гибкой – ее функции можно наращивать, модифицировать или сужать, добавляя, модифицируя или исключая серверы пользовательского режима. Кроме того, серверы хорошо защищены друг от друга, как и любые пользовательские процессы.

3. КОМАНДНЫЙ ИНТЕРФЕЙС ОС WINDOWS

В операционной системе Windows, интерактивные команды выполняются командным процессором (command shell). Для запуска командного интерпретатора (открытия командной строки) необходимо выбрать пункт «**Выполнить**…» (**Run**) стартового меню, ввести имя файла командного процессора – **Cmd.exe**.

3.1. Синтаксис командной строки, управление вводом – выводом

Файловая система имеет древовидную структуру и имена файлов задаются в формате: *[диск:] [путь\]имя_файла*. Если путь начинается с символа «\», то маршрут вычисляется от корневого каталога – иначе от текущего.

Существуют особые обозначения для текущего каталога (точка «.») и трех его верхних уровней (две точки «..» – родительский, три «…» – второго уровня и, наконец, четыре «….» – третьего уровня).

Спецсимволы управления выполнением команд: «**>**» – перенаправления вывода на стандартное устройство PRN, COM1, и т. п или файл; «**>>**» – позволяет дописать в уже существующий файл; «**<**» – ввод данных не с клавиатуры, а с файла; **команда1 | команда2** – конвейеризация команд, когда сообщения, выводимые первой командой, используются в качестве входных данных для второй.

Спецсимволы управления группировкой команд: **команда1 && команда2** – группировка команд, выполнение команды2 только в случае успешного выполнения конманды1; **команда1 || команда2** – выполнение команды2 только при не выполнении команды1.

Примеры

1. *Вывод встроенной справки для команды COPY в файл copy.txt:* ***COPY /? > copy.txt***

2. *Добавление текста справки для команды XCOPY в файл copy.txt:* ***XCOPY /? >> copy.txt***

3. *Ввод новой даты из файла:* ***DATE < date.txt***

4. *При отсутствии файла «C:\plan.txt» будет выведено на экран содержимое папки*:

TYPE C:\plan.txt || DIR

3.2. Переменные окружения

При загрузке ОС Windows в оперативной памяти постоянно хранится набор *переменных окружения* (environment variables). Переменные устанавливаются с помощью команды:

SET переменная = значение

Запуск **SET** без параметров приводит к выводу списка переменных среды. Для получения их значений необходимо имя соответствующей переменной заключить в символы «**%**», например: **%TEMP%** или **%PATH%**.

3.3. Основные команды

Полный список команд можно вывести, набрав **HELP** в командной строке. Для вызова справки к отдельной команде необходимо запустить ее с ключом **/?**. Остановимся подробнее на командах работы с файловой системой: ATTRIB, CD, COPY, XCOPY, DIR, MKDIR, RMDIR, DEL, REN, MOVE.

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

Используется для просмотра или установки атрибутов файлов: Read–Only (**R**) – только для чтения; System (**S**) – системный; Archive (**A**) – архивный; Hidden (**H**) – скрытый. Установка атрибута производится ключом «**+**», снятие – «**-**». Ключ «**/S**» применяется для обработки файлов во всех подкаталогах указанного пути. Ключ «**/D**» – обработка и файлов, и папок.

Например, сделать все Word-файлы в каталоге «Мои документы» доступными только для чтения: ***ATTRIB +R "C:\Мои документы*.doc***

Команда **CD** [диск:][путь\]

Смена текущей папки. Например, команда **CD ** выполняет переход в корневой каталог текущего диска. Если запустить команду **CD** без параметров, то на экран будут выведены имена текущего диска и каталога.

Команда **COPY** [/D] [/V] [/N] [/Y | /-Y] [/Z] [/A | /B] источник [/A | /B][+ источник [/A | /B] [+ ...]] [результат [/A | /B]]

Команда предназначена для копирования одного или нескольких файлов. Приведем примеры использования команды **COPY**. Атрибуты: «источник» – имена одного или нескольких копируемых файлов; «результат» – папка и/или имя для конечных файлов; **/A** – файл является текстовым файлом ASCII; **/B** – файл является двоичным файлом; **/V** – проверка правильности копирования файлов; **/Y** – подавление запроса подтверждения на перезапись.

Примеры

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

COPY abc.txt D:\PROGRAM

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

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

3. ***Мои документы**» на диске **C** (имя с пробелами берется в кавычки): **COPY A:*.txt "C:\Мои документы"***

4. *Печать файла **abc.txt**: **COPY abc.txt PRN***

5. *Создание нового текстового файла (**Примечание**. Для завершения ввода необходимо ввести символ конца файла – <Ctrl>+<Z>): COPY CON my.txt*

Команда **COPY** может также объединять нескольких файлов в один. Это достигается использованием символов группировки «**?**» и «*****» или знака конткатенации «**+**».

Примеры

1. *Объединить файлы **1.txt** и **2.txt** в файл **3.txt**: **COPY 1.txt+2.txt 3.txt***

2. *Объединить все файлы с расширением **dat** из текущего каталога в один файл* all.dat *(**Примечание**. Ключ /B используется для предотвращения усечения соединяемых файлов, поскольку при комбинировании файлов команда **COPY** по умолчанию считает файлы текстовыми.): **COPY /B *.dat all.dat***

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

Команда **XCOPY** источник [результат] [ключи]

Свободна от недостатков предыдущей команды, но предназначена только для копирования файлов и папок, не работает с устройствами. Команда **XCOPY** имеет множество ключей, рассмотрим основные. Ключ **/D[:[дата]]** позволяет копировать только файлы, измененные не ранее указанной даты. Если параметр дата не указан, то копирование будет производиться, только если источник новее результата. Ключ **/S** позволяет копировать все непустые папки в каталоге-источнике. С помощью же ключа **/E** можно копировать вообще все подкаталоги, включая и пустые. Ключ **/C**, копирование будет продолжаться даже в случае возникновения ошибок. При задании ключа **/Q** имена файлов при копировании не отображаются, ключа **/F** – отображаются полные пути источника и результата. С помощью ключа **/H** можно копировать скрытые и системные файлы, а с помощью ключа **/R** – заменять файлы с атрибутом "Только для чтения". Ключ

/T позволяет применять **XCOPY** для копирования только структуры каталогов источника. Для того чтобы копировать не только данные, но и полностью атрибуты файла, необходимо использовать ключ **/K**.

Примеры

1. *Скопировать в каталог **'D:\BACKUP\Мои документы'** только измененные файлы из каталога **'C:\Мои документы'**:*

XCOPY "C:\Мои документы*.*" "D:\BACKUP\Мои документы" /D

2. *Скопировать все файлы из корневого каталога диска C: (включая системные и скрытые) в каталог SYS на диске D:*

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

Команда **DIR** [диск:][путь][имя_файла] [ключи]

Используется для вывода информации о содержимом дисков и каталогов. Если задать эту команду без параметров, то выводится содержимое текущего каталога. С помощью ключа **/A[[:]атрибуты]** можно вывести имена только тех каталогов и файлов, которые имеют заданные атрибуты (**R** – "Только чтение", **A** – "Архивный", **S** – "Системный", **H** – "Скрытый"). Ключ **/S** вывод списка файлов из заданного каталога и его подкаталогов.

Примеры

1. *Вывод на экран всего содержимого папки **'C:\Мои документы'**:*

DIR "C:\Мои документы"

2. *Вывод на экран всех файлов с расширением bat в корневом каталоге диска C*: ***DIR C:*.bat***

3. *Вывод имен всех файлов в корневом каталоге диска C:, которые одновременно являются скрытыми и системными*:

DIR C:\ /A:HS

4. *Для вывода всех файлов, кроме скрытых:* ***DIR C:\ /A:-H***

5. *Вывод списка всех каталогов диска C:* ***DIR C: /A:D***

Команды **MKDIR** [диск:] путь и **RMDIR** [диск:]путь [ключи]

Для создания нового каталога и удаления уже существующего пустого каталога используются команды MKDIR и RMDIR соответственно (или их короткие аналоги MD и RD). Ключ **/S** позволяет задавать удаление дерева каталогов.

Примеры

1. *Создание папки:* ***MD "C:\Примеры"***

2. *Удаление папки со вложенными каталогами:*

RD "C:\Примеры" /S

Команда **DEL** [диск:][путь]имя_файла [ключи]

Предназначена для удаления одного или группы файлов. Для удаления сразу нескольких файлов используются групповые знаки **?** и *****. Ключ **/S** позволяет удалить указанные файлы из всех подкаталогов, ключ **/F** – принудительно удалить файлы, доступные только для чтения, ключ **/A[[:]атрибуты]** – отбирать файлы для удаления по атрибутам.

Пример

1. ***Удалить все файлы в текущей папке: DEL ****

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

Предназначена для переименования файлов и каталогов. Здесь параметр «**каталог1|файл1»** определяет название, которое нужно изменить, а «**каталог2|файл2»** задает новое. В любом параметре команды **REN** можно использовать групповые символы **?** и *****.

Пример

1. *В текущей директории все файлы с расширением* ***txt*** *заменить на расширение* ***doc****:* ***REN *.txt *.doc***

Команда **MOVE** [/Y|/–Y] [диск:][путь]имя_файла1[,...] итоговый_файл

Команда предназначена для перемещения одного или более файлов. Параметр «**итоговый_файл**» задает новое размещение файла и может включать имя диска, двоеточие, имя каталога, либо их сочетание. Если перемещается только один файл, допускается указать его новое имя. Это позволяет сразу переместить и переименовать файл.

Пример

1. *Перенести файл «список.txt» на диск D с переименованием в «list.txt»:*

MOVE "C:\Мои документы\список.txt" D:\list.txt

3.4. Язык интерпретатора Cmd.exe. Командные файлы

Язык оболочки командной строки (shell language) в Windows реализован в виде командных (или пакетных) файлов. Командный файл в Windows – это обычный текстовый файл с расширением **bat** или **cmd**, в котором записаны команды операционной системы, а также некоторые дополнительные инструкции и ключевые слова. Для прерывания выполнения командного файла необходимо нажать **<CTRL> +<C>**.

3.4.1. Вывод сообщений

Основной командой для вывода информации в пакетных файлах служит: **ECHO [сообщение]**.

Команда может работать с механизмом перенаправления вывода. По умолчанию команды пакетного файла перед исполнением выводятся на экран, что не очень эстетично. С помощью команды **ECHO OFF** можно отключить дублирование вывода команд на экран. **ECHO ON** восстанавливает режим дублирования. Кроме этого, можно отключить дублирование отдельной строки, если предварить ее символом «@». Часто бывает удобно для просмотра сообщений, выводимых из пакетного файла, предварительно полностью очистить экран командой **CLS**.

Пример *использования команды* **ECHO**:

```
CLS
REM Следующая команда не будет дублироваться на экране
@DIR E:\
REM Следующие две команды будут дублироваться на экране
DIR C:\
ECHO OFF
REM А остальные уже не будут
DIR D:\
PAUSE
```

3.4.2. Использование параметров командной строки

При запуске пакетных файлов в командной строке можно указывать произвольное число параметров, значения которых можно использовать внутри файла. Для доступа из командного файла к параметрам командной строки применяются символы **%0**, **%1**, …, **%9** или **%***. При этом вместо **%0** подставляется имя выполняемого пакетного файла, вместо **%1**, **%2**, …, **%9** – значения первых девяти параметров командной строки соответственно, а вместо **%*** – все аргументы.

Пример

*Реализовать командный файл «**copier.bat**» для копирования файлов и папок из указанного места в заданное. Параметры передаются из командной строки.*

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

При необходимости можно использовать более девяти параметров командной строки. Это достигается с помощью команды **SHIFT**, которая изменяет значения замещаемых параметров с **%0** по **%9**, копируя каждый параметр в предыдущий, т. е. значение **%1** помещается в **%0**, значение **%2** – в **%1** и т. д. При включении расширенной обработки команд **SHIFT** поддерживает ключ **/n**, задающий начало сдвига параметров с номера **n**, где **n** может быть числом от 0 до 9.

Например, в следующей команде: **SHIFT /2** параметр **%2** заменяется на **%3**, **%3** на **%4** и т. д., а параметры **%0** и **%1** остаются без изменений. При работе с параметрами командного файла можно использовать синтаксический анализ (табл. 3.1):

Таблица 3.1

Операторы	Описание	Пример
%~Fn	Полное имя файла	%~F1=C:\TEXT\Рассказ.doc
%~Dn	Имя диска	%~D1=C:
%~Pn	Путь к файлу	%~P1=\TEXT\
%~Nn	Имя файла	%~N1=Рассказ
%~Xn	Расширение файла	%~X1=doc

3.4.3. Работа с переменными среды

В командных файлах можно использовать переменные окружения и объявлять собственные с помощью команды **SET**. Для получения значения определенной переменной среды нужно имя этой переменной заключить в символы **%**.

Пример

Вывести значение системное переменной **WinDir**.

```
@ECHO OFF
CLS
ECHO Значение переменной WinDir: %WinDir%
```

С переменными среды в командных файлах можно производить операцию конкатенации. Не следует для конкатенации использовать знак +, так как он будет воспринят просто в качестве символа. Например:

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

После выполнения значением **C** будет являться строка '**РазДва**', а **D – Раз+Два**.

Значения переменных среды можно рассматривать и как числовые и производить с ними арифметические вычисления. Для этого используется команда **SET** с ключом **/A**.

Пример

*Реализовать командный файл «**add.bat**» для суммирования двух чисел, заданных в качестве параметров командной строки*:

```
@ECHO OFF
REM В переменной M будет храниться сумма
SET /A M=%1+%2
ECHO Сумма %1 и %2 равна %M%
```

3.4.5. Операторы перехода

Командный файл может содержать метки, начинающиеся с двоеточия «**:**» и команды **GOTO** перехода к этим меткам. Имя метки задается набором символов, следующих за двоеточием. Приведем пример.

```
@ECHO OFF
GOTO Label1
ECHO Эта строка никогда не выполнится
:Label1
REM Продолжение выполнения
DIR
```

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

3.4.6. Операторы условия

С помощью команды **IF … ELSE** (**ELSE** может отсутствовать) в пакетных файлах можно выполнять обработку условий нескольких типов.

Проверка значения переменной

Используется для проверки значения переменной. Варианты синтаксиса команды (квадратные скобки указывают на необязательность параметра):

```
IF [NOT] строка1==строка2 команда1 [ELSE команда2]
IF [/I] [NOT] строка1 оператор_сравнения строка2 команда
```

Ключ **/I** задает сравнение текстовых строк без учета регистра. Строки могут быть литеральными или представлять собой значения переменных (например, **%1** или **%TEMP%**). Кавычки для литеральных строк не требуются. Например,

IF %1==%2 ECHO Параметры совпадают!
IF %1==Петя ECHO Привет, Петя!

При сравнении строк, заданных переменными, следует проявлять осторожность, поскольку значение может оказаться пустым, и тогда возникнет ситуация, при которой выполнение командного файла аварийно завершится. Например, если переменная **MyVar** не определена, а в файле имеется условный оператор вида:

IF %MyVar%==C:\ ECHO Ура!!!

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

IF -%MyVar%==-C:\ ECHO Ура!!!

С помощью команд **IF** и **SHIFT** можно в цикле обрабатывать все параметры командной строки файла, даже не зная заранее их количества.

Пример

*Реализовать командный файл «**all.bat**» для вывода на экран имени запускаемого файла и всех параметров командной строки:*

```
@ECHO OFF
ECHO Выполняется файл: %0
ECHO.
ECHO Файл запущен со следующими параметрами...
REM Начало цикла
:BegLoop
IF -%1==- GOTO ExitLoop
ECHO %1
REM Сдвиг параметров
SHIFT
REM Переход на начало цикла
GOTO BegLoop
```

:ExitLoop
REM Выход из цикла
ECHO.
ECHO Все.

Возможные значения **операторов_сравнения**: **EQL** – Равно; **NEQ** – Не равно; **LSS** – Меньше; LEQ – Меньше или равно; **GTR** – Больше; **GEQ** – Больше или равно.

IF -%1 EQL -Вася ECHO Привет, Вася!
IF -%1 NEQ -Вася ECHO Привет, но Вы не Вася!

Проверка существования заданного файла

Второй способ использования команды **IF** – это проверка существования заданного файла. Синтаксис:

IF [NOT] EXIST файл команда1 [ELSE команда2]

Условие считается истинным, если указанный файл существует. Кавычки для имени файла не требуются. Приведем пример проверки наличия заданного файла:

IF NOT EXIST C:\autoexec.bat ECHO У вас нет файла автозагрузки

IF EXIST "C:\Мои документы\Работа.doc" ECHO Все в порядке

Проверка наличия переменной среды

Для проверки наличия переменной среды используется синтаксис команды **IF** в виде:

IF DEFINED переменная команда1 [ELSE команда2]

Пример

Приведенный командный файл выполняет проверку наличия системной переменной **MyVar**.

@ECHO OFF
CLS
IF DEFINED MyVar GOTO :VarExists
ECHO Переменная MyVar не определена
GOTO :EOF
:VarExists
ECHO Переменная MyVar определена,
ECHO ее значение равно %MyVar%

Проверка кода завершения предыдущей команды

Еще один способ использования команды **IF** – проверка кода завершения (кода выхода) предыдущей команды. Условие считается истинным, если последняя запущенная команда или программа завершилась с кодом возврата, равным, либо превышающим указанное число. Синтаксис:

IF [NOT] ERRORLEVEL число команда1 [ELSE команда2]

В операторе можно также применять операторы сравнения.

Пример

Выполнить копирование файла ***my.txt*** *на диск* ***C:*** *без вывода на экран сообщений о копировании, а в случае возникновения какой-либо ошибки выдавать предупреждение*.

```
@ECHO OFF
XCOPY my.txt C:\ > NUL
REM Проверка кода завершения копирования
IF ERRORLEVEL 1 GOTO ErrOccurred
ECHO Копирование выполнено без ошибок.
GOTO :EOF

:ErrOccurred
ECHO При выполнении команды XCOPY возникла ошибка!
```

3.4.7. Организация циклов

В командных файлах для организации циклов используются несколько разновидностей оператора FOR.

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

Выполнение заданной команды для каждого элемента множества. Например:

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

напечатает следующее: ***Раз Два Три***

В качестве переменных цикла можно использовать лишь имена, состоящие из одной буквы. В параметре **множество** можно представить одну или несколько групп файлов. Например, для вывода всех файлов с расширениями *.doc и *.txt:

FOR %%f IN(D:*.doc D:*. txt) DO ECHO %%f >>list.txt

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

Выполнение заданной команды для всех подходящих имен каталогов.

Пример

Получить список всех каталогов диска **C**:

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

FOR /R [[диск:]путь] %%переменная IN (набор) DO команда [параметры]

Организует рекурсивное выполнение заданной команды для определенного каталога, а также всех его подкаталогов. Если вместо набора указана только точка «**.**», то команда проверяет все подкаталоги текущего каталога.

Пример

Получить список всех каталогов и подкаталогов в текущей папке:

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

Пример

Получить список имен всех файлов с расширением ****.txt****:*

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

FOR /L %%переменная IN (начало, шаг, конец) DO команда [параметры]

Цикл с параметром, заданный началом, концом и шагом приращения.

Пример

Задать цикл с изменением параметра от 1 до 5 с шагом 1.

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

Числа, получаемые в результате выполнения цикла **FOR /L**, можно использовать в арифметических вычислениях. Рассмотрим работу командного файла «**my.bat»** следующего содержания:

```
@ECHO OFF
CLS
FOR /L %%f IN (1,1,5) DO CALL :2 %%f
GOTO :EOF
:2
SET /A M=10*%1
ECHO 10*%1=%M%
```

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

10*1=10

10*5=50

FOR /F [“ключи”] %%переменная IN (набор) DO команда [параметры]

Обработка состоит в чтении файла, разбиении его на отдельные строки текста и выделении из каждой строки заданного числа подстрок. Затем найденная подстрока используется в качестве значения переменной при выполнении основного тела цикла. Ключи позволяют выделять слова и подстроки (токены) по разделителям (DELIMS) или начальным символам.

3.5. Пример

*Написать командный файл, который будет копировать из текущего каталога все файлы с расширением **txt**, кроме одного файла, указанного в качестве второго параметра командной строки, в каталог, указанный первым параметром. Если имя каталога, в который должно производиться копирование, не задано, то вывести сообщение об этом и прервать выполнение файла*.

Решение

*Для выполнения поставленной задачи можно перебрать в цикле все файлы с расширением **txt**, проверяя перед копированием имя каждого из этих файлов*:

```
@ECHO OFF
REM Проверка наличия параметра командной строки
IF -%!==- GOTO NoDir
REM Копирование нужных файлов
FOR %%f IN (*.txt) DO IF NOT -%%f==-%2 COPY %%f %1
GOTO End
:NoDir
ECHO Не указан каталог для копирования!
PAUSE
:End
```

4. СЕРВЕР СЦЕНАРИЕВ WINDOWS SCRIPT HOST

Windows Script Host (WSH) предназначен для выполнения сценариев, поддерживающих технологию ActiveX Scripting. В качестве стандартных языков поддерживаются Visual Basic Script Edition (VBScript) и JScript. WSH является удобным инструментом для автоматизации повседневных задач пользователей и администраторов операционной системы Windows. Используя WSH, можно работать с файловой системой компьютера, а также управлять работой других приложений. WSH – сценарии представляют собой текстовые файлы с расширением **js** или **vbs**.

4.1. Запуск сценария из командной строки в консольном режиме

Сценарий можно выполнить в двух режимах: консольном и графическом. Например:

cscript C:\MyScript.vbs – консольный режим
wscript C:\MyScript.vbs – графический режим

Если сценарий запускается в графическом режиме, то его свойства можно устанавливать с помощью вкладки **Сценарий** (**Script**) диалогового окна, задающего свойства файла в Windows.

4.2. Собственная объектная модель WSH

Объектная модель WSH версии 5.6 состоит из следующих объектов:

1. **WScript** – главный объект, который служит для создания других объектов или связи с ними, содержит сведения о сервере сценариев, а также позволяет вводить данные с клавиатуры и выводить информацию на экран или в окно Windows.
2. **WshArguments** – обеспечивает доступ ко всем параметрам командной строки запущенного сценария или ярлыка Windows.
3. **WshNamed** – обеспечивает доступ к именным параметрам командной строки запущенного сценария.
4. **WshUnnamed** – обеспечивает доступ к безымянным параметрам командной строки запущенного сценария.
5. **WshShell** – позволяет запускать независимые процессы, создавать ярлыки, работать с переменными среды, системным реестром и специальными папками Windows.

6. **WshSpecialFolders** – обеспечивает доступ к специальным папкам Windows.

7. **WshShortcut** – позволяет работать с ярлыками Windows.

8. **WshUrlShortcut** – предназначен для работы с ярлыками сетевых ресурсов.

9. **WshEnvironment** – предназначен для просмотра, изменения и удаления переменных среды.

10. **WshNetwork** – используется при работе с локальной сетью: содержит сетевую информацию для локального компьютера, позволяет подключать сетевые диски и принтеры.

11. **WshScriptExec** – позволяет запускать консольные приложения в качестве дочерних процессов, обеспечивает контроль состояния этих приложений и доступ к их стандартным входным и выходным потокам.

12. **WshController** – позволяет запускать сценарии на удаленных машинах.

13. **WshRemote** – позволяет управлять сценарием, запущенным на удаленной машине.

14. **WshRemoteError** – используется для получения информации об ошибке, возникшей в результате выполнения сценария, запущенного на удаленной машине.

15. **FileSystemObject** – обеспечивающий доступ к файловой системе компьютера.

4.3. Объект WScript

В сценарии WSH объект **WScript** можно использовать без предварительного описания или создания, так как его экземпляр создается автоматически.

Таблица 4.1

Основные свойства объекта WScript

Свойство	Описание
Arguments	Содержит указатель на коллекцию WshArguments, содержащую параметры командной строки для исполняемого сценария
ScriptFullName	Содержит полный путь к запущенному сценарию
ScriptName	Содержит имя запущенного сценария

Окончание табл. 4.1

Свойство	Описание
StdErr	Позволяет записывать сообщения в стандартный поток для ошибок
StdIn	Позволяет читать информацию из стандартного входного потока
StdOut	Позволяет записывать информацию в стандартный выходной поток

Таблица 4.2

Основные методы объекта WScript

Метод	Описание
CreateObject(strProgID [, strPrefix])	Создает объект, заданный параметром strProgID
Echo([Arg1] [, Arg2] [,…])	Выводит текстовую информацию на консоль или в диалоговое окно
Quit([intErrorCode])	Прерывает выполнение сценария с заданным параметром intErrorCode кодом выхода. Если параметр intErrorCode не задан, то объект WScript установит код выхода, равным нулю
Sleep(intTime)	Приостанавливает выполнение сценария на заданное число миллисекунд

Для использования же всех остальных объектов нужно использовать метод **CreateObject**. Основные свойства и методы объекта **WScript** представлены в табл. 4.1 и 4.2 соответственно.

Свойство Arguments

Для перебора коллекции параметров командной строки, можно воспользоваться циклом For Each … Next.

```
Dim i,objArgs,Arg
Set objArgs = WScript.Arguments ' Создаем объект WshArguments
For Each Arg In objArgs
  WScript.Echo Arg ' Вывод значения аргумента
Next
```

Свойства StdErr, StdIn, StdOut

Доступ к стандартным входным и выходным потокам можно получить только в случае, если сценарий запущен в консольном режиме.

Таблица 4.3

Основные методы для работы с потоками

Метод	Описание
Read(n)	Считывает из потока StdIn заданное параметром n число символов и возвращает полученную строку
ReadAll()	Читает символы из потока StdIn до тех пор, пока не встретится символ конца файла (<Ctrl>+<Z>)
ReadLine()	Возвращает строку, считанную из потока StdIn
Skip(n)	Пропускает при чтении из потока StdIn заданное параметром n число символов
SkipLine()	Пропускает целую строку при чтении из потока StdIn
Write(string)	Записывает в поток StdOut или StdErr строку string (без символа конца строки)
WriteBlankLines(n)	Записывает в поток StdOut или StdErr заданное параметром n число пустых строк
WriteLine(string)	Записывает в поток StdOut или StdErr строку string (вместе с символом конца строки)

Пример

```
Dim  s
' Выводим строку на экран
WScript.StdOut.Write "Введите число: "
' Считываем строку
s = WScript.StdIn.ReadLine
' Выводим строку на экран
WScript.StdOut.WriteLine "Вы ввели число " & s
```

Метод CreateObject

Строковый параметр **strProgID**, указываемый в методе **CreateObject**, называется программным идентификатором объекта. Пример создания объекта **WshNetwork:**

```
set WshNetwork = WScript.CreateObject("WScript.Network")
```

Метод Echo

Параметры **Arg1**, **Arg2**, … метода **Echo** задают аргументы для вывода. Если сценарий был запущен с помощью **wscript**, то метод **Echo** направляет вывод в диалоговое окно, если же для выполнения сценария применяется cscript.exe, то вывод будет направлен на консоль.

WScript.Echo 1,2,3 'Выводим числа

WScript.Echo "Привет!" 'Выводим строку

4.4. Объект WshShell

С помощью объекта **WshShell** можно запускать новый процесс, создавать ярлыки, работать с системным реестром, получать доступ к переменным среды и специальным папкам Windows. Создается этот объект следующим образом:

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

Таблица 4.4

Основные свойства объекта WshShell

Свойство	Описание
CurrentDirectory	Здесь хранится полный путь к текущему каталогу (к каталогу, из которого был запущен сценарий)
Environment	Содержит объект WshEnvironment, который обеспечивает доступ к переменным среды операционной системы
SpecialFolders	Содержит объект WshSpecialFolders для доступа к специальным папкам Windows (рабочий стол, меню Пуск (Start) и т. д.)

Таблица 4.5

Основные методы объекта WshShell

Метод	Описание
AppActivate (title)	Активизирует заданное параметром title окно приложения. Строка title задает название окна
CreateShortcut (strPathname)	Создает объект WshShortcut для связи с ярлыком Windows или объект WshUrlShortcut для связи с сетевым ярлыком. Параметр strPathname задает полный путь к создаваемому или изменяемому ярлыку
Environment (strType)	Возвращает объект WshEnvironment, содержащий переменные среды заданного вида ("System" – переменные среды операционной системы), "User" – переменные среды пользователя, "Volatile" – временные переменные, "Process" – переменные среды текущего командного окна)

Окончание табл. 4.5

Метод	Описание
Exec (strCommand)	Создает новый дочерний процесс, который запускает консольное приложение, заданное параметром strCommand
ExpandEnviron-mentStrings (strString)	Возвращает значение переменной среды текущего командного окна, заданной строкой strString (имя переменной должно быть окружено знаками "%")
Popup (strText,[nSecToW ait], [strTitle], [nType])	Выводит на экран информационное окно с сообщением, заданным параметром strText. Параметр nSecToWait задает количество секунд, по истечении которых окно будет автоматически закрыто, параметр strTitle определяет заголовок окна, параметр nType указывает тип кнопок и значка для окна
RegDelete (strName)	Удаляет из системного реестра заданный параметр или раздел целиком
RegRead (strName)	Возвращает значение параметра реестра или значение по умолчанию для раздела реестра
RegWrite (strName, any-Value [,strType])	Записывает в реестр значение заданного параметра или значение по умолчанию для раздела
Run (strCommand, [intWindowStyle], [bWaitOnReturn])	Создает новый независимый процесс, который запускает приложение, заданное параметром strCommand
SendKeys (string)	Посылает одно или несколько нажатий клавиш в активное окно (эффект тот же, как если бы вы нажимали эти клавиши на клавиатуре)
SpecialFolders (strSpecFolder)	Возвращает строку, содержащую путь к специальной папке Windows, заданной параметром strSpecFolder

Пример сценария для создания двух ярлыков – на сам выполняемый сценарий (объект **oShellLink**) и на сетевой ресурс (**oUrlLink**).

```
Dim WshShell,oShellLink,oUrlLink
' Создаем объект WshShell
Set WshShell=WScript.CreateObject("WScript.Shell")
' Создаем ярлык на файл
Set oShellLink=WshShell.CreateShortcut("Current Script.lnk")
' Устанавливаем путь к файлу
oShellLink.TargetPath=WScript.ScriptFullName
' Сохраняем ярлык
oShellLink.Save
```

```
' Создаем ярлык на сетевой ресурс
Set oUrlLink = WshShell.CreateShortcut("GSTU.URL")
' Устанавливаем URL
oUrlLink.TargetPath = "http://gstu.local"
' Сохраняем ярлык
oUrlLink.Save
```

Пример сценария для вывода некоторых переменных среды.

```
Dim WshShell,WshSysEnv
' Создаем объект WshShell
Set WshShell = WScript.CreateObject("WScript.Shell")
' Создание коллекции WshEnvironment
Set WshSysEnv = WshShell.Environment("SYSTEM")
WScript.Echo WshSysEnv("OS")
WScript.Echo
WshShell.Environment.Item("NUMBER_OF_PROCESSORS")
```

Пример сценария для вывода кода выхода запущенного приложения

```
' Создаем объект WshShell
Set WshShell = WScript.CreateObject("WScript.Shell")
' Запускаем Блокнот и ожидаем завершения его работы
Return = WshShell.Run("notepad " + WScript.ScriptFullName, 1,
true)
' Печатаем код возврата
WScript.Echo "Код возврата:", Return
```

4.5. Объект WshEnvironment

Объект **WshEnvironment** позволяет получить доступ к коллекции, содержащей переменные среды заданного типа. Объект имеет свойство **Length**, в котором хранится число элементов в коллекции, и методы **Count** и **Item**. Для того, чтобы получить значение определенной переменной среды, в качестве аргумента метода **Item** указывается имя этой переменной в двойных кавычках.

Пример сценария вывода значения с переменной среды *PATH*.

```
Dim WshShell, WshSysEnv
Set WshShell=WScript.CreateObject("WScript.Shell")
Set WshSysEnv=WshShell.Environment
WScript.Echo "Системный путь:",WshSysEnv.Item("PATH")
```

4.6. Объект WshSpecialFolders

Объект **WshSpecialFolders** обеспечивает доступ к коллекции, содержащей пути к специальным папкам Windows. В Windows поддерживаются следующие имена специальных папок: Desktop; Favorites; Fonts; MyDocuments; NetHood; PrintHood; Programs; Recent; SendTo; StartMenu; Startup; Templates; AllUsersDesktop; AllUsersStartMenu; AllUsersPrograms; AllUsersStartup.

Пример сценария, формирующего список всех имеющихся в системе специальных папок.

```
Dim WshShell, WshFldrs, SpecFldr, s  ' Объявляем переменные
' Создаем объект WshShell
Set WshShell = WScript.CreateObject(''Wscript.Shell'')
' Создаем объект WshSpecialFolders
Set WshFldrs = WshShell.SpecialFolders
s=''Список всех специальных папок:'' & vbCrLf & vbCrLf
' Перебираем все элементы коллекции WshFldrs
For Each SpecFldr In WshFldrs
  ' Формируем строки с путями к специальным папкам
  s=s & SpecFldr & vbCrLf
Next
WScript.Echo s
```

Пример сценария для доступа к заданной специальной папке.

```
Dim WshShell, WshFldrs, s   ' Объявляем переменные
' Создаем объект WshShell
Set WshShell = WScript.CreateObject(''Wscript.Shell'')
' Создаем объект WshSpecialFolders
Set WshFldrs = WshShell.SpecialFolders
' Формируем строки с путями к конкретным специальным папкам
s=''Некоторые специальные папки:'' & vbCrLf & vbCrLf
s=s+''Desktop:''+WshFldrs(''Desktop'') & vbCrLf
s=s+''Favorites:''+WshFldrs(''Favorites'') & vbCrLf
s=s+''Programs:''+WshFldrs(''Programs'')
WScript.Echo s   ' Выводим сформированные строки на экран
```

4.7. Объектная модель FileSystemObject

С помощью методов объекта **FileSystemObject** можно выполнять все основные действия с папками и файлами, читать информацию из текстовых файлов и записывать в них данные. В таблице 4.6 приведены основные объекты для работы с файловой системой.

Таблица 4.6

Основные объекты для работы с файловой системой

Операция	Используемые объекты, свойства и методы
Получение сведений об определенном диске (файловая система, метка тома, объем и т. д.)	Свойства объекта Drive. Объект создается с помощью метода GetDrive объекта FileSystemObject
Получение сведений о заданном каталоге или файле (дата создания, размер, атрибуты и т. д.)	Свойства объектов Folder и File. Объекты создаются с помощью методов GetFolder и GetFile объекта FileSystemObject
Проверка существования определенного диска, каталога или файла	Методы DriveExists, FolderExists и FileExists объекта FileSystemObject
Копирование файлов и каталогов	Методы CopyFile и CopyFolder объекта FileSystemObject а также методы File.Copy и Folder.Copy
Перемещение файлов и каталогов	Методы MoveFile и MoveFolder объекта FileSystemObject или методы File.Move и Folder.Move
Удаление файлов и каталогов	Методы DeleteFile и DeleteFolder объекта FileSystemObject или методы File.Delete и Folder.Delete
Создание каталога	Методы FileSystemObject.CreateFolder или Folders.Add
Создание текстового файла	Методы FileSystemObject.CreateTextFile или Folder.CreateTextFile
Получение списка всех доступных дисков	Коллекция Drives, содержащаяся в свойстве FileSystemObject.Drives
Получение списка всех подкаталогов заданного каталога	Коллекция Folders, содержащаяся в свойстве Folder.SubFolders
Получение списка всех файлов заданного каталога	Коллекция Files, содержащаяся в свойстве Folder.Files

Окончание табл. 4.6

Открытие текстового файла для чтения, записи или добавления	Методы FileSystemObject.CreateTextFile или File.OpenAsTextStream
Чтение информации из заданного текстового файла или запись ее в него	Методы объекта TextStream

Пример сценария для вывода на экран свойств диска С.

```
'Объявляем переменные
Dim FSO,D,TotalSize,FreeSpace,s
'Создаем объект FileSystemObject
Set FSO = WScript.CreateObject("Scripting.FileSystemObject")
'Создаем объект Drive для диска C
Set D = FSO.GetDrive("C:")
s = "Информация о диске C:" & VbCrLf
'Получаем серийный номер диска
s = s &"Серийный номер: " & D.SerialNumber & VbCrLf
'Получаем метку тома диска
s = s & "Метка тома: " & D.VolumeName & VbCrLf
'Вычисляем общий объем диска в килобайтах
TotalSize = D.TotalSize/1024
s = s & "Объем: " & TotalSize & " Kb" & VbCrLf
'Вычисляем объем свободного пространства диска в килобайтах
FreeSpace = D.FreeSpace/1024
s = s & "Свободно: " & FreeSpace & " Kb" & VbCrLf
'Выводим свойства диска на экран
WScript.Echo s
```

Пример сценария для вывода свойств текущего каталога.

```
Dim FSO,WshShell,FoldSize,s  'Объявляем переменные

'Создаем объект FileSystemObject
Set FSO = WScript.CreateObject("Scripting.FileSystemObject")
'Создаем объект WshShell
Set WshShell = WScript.CreateObject("WScript.Shell")

'Определяем каталог, из которого был запущен сценарий
'(текущий каталог)
Set Folder = FSO.GetFolder(WshShell.CurrentDirectory)
'Получаем имя текущего каталога
```

```
s = "Текущий каталог: " & Folder.Name & VbCrLf
'Получаем дату создания текущего каталога
s = s & "Дата создания: " & Folder.DateCreated & VbCrLf
'Вычисляем размер текущего каталога в килобайтах
FoldSize=Folder.Size/1024
s = s & "Объем: " & FoldSize & " Kb" & VbCrLf
'Выводим информацию на экран
WScript.Echo s
```

Пример сценария для проверки существования заданного файла.

```
Dim FSO,FileName 'Объявляем переменные

'Создаем объект FileSystemObject
Set FSO = WScript.CreateObject("Scripting.FileSystemObject")

FileName = "C:\boot.ini"
if  FSO.FileExists(FileName)  Then
'Выводим информацию на экран
WScript.Echo "Файл " & FileName & " существует"
else
'Выводим информацию на экран
WScript.Echo "Файл " & FileName & " не существует"
end if
```

Пример сценария для вывода на экран сведения о всех доступных дисках.

```
'Объявляем переменные
Dim FSO,s,ss,Drives,D

'Создаем объект FileSystemObject
Set FSO = WScript.CreateObject("Scripting.FileSystemObject")
'Создаем коллекцию дисков, имеющихся в системе
Set Drives = FSO.Drives
s = ""

'Перебираем все диски в коллекции
For Each D In Drives
'Получаем букву диска
s = s & D.DriveLetter
s = s & " - "
if (D.DriveType = 3) then  'Проверяем, не является ли диск сетевым
```

```
'Получаем имя сетевого ресурса
ss = D.ShareName
else
'Диск является локальным
if (D.IsReady) then  'Проверяем готовность диска
'Если диск готов, то получаем метку тома для диска
ss = D.VolumeName
else
ss = "Устройство не готово"
end if
s = s & ss & VbCrLf
end if
Next

'Выводим полученные строки на экран
WScript.Echo s
```

Пример сценария для вывода на экран названия всех файлов, которые содержатся в специальной папке «Мои документы».

```
'Объявляем переменные
Dim FSO,F,File,Files,WshShell,PathList,s
'Создаем объект FileSystemObject
Set FSO = WScript.CreateObject("Scripting.FileSystemObject")
'Создаем объект WshShell
Set WshShell = WScript.CreateObject("Wscript.Shell")
'Создаем объект WshSpecialFolders
Set WshFldrs = WshShell.SpecialFolders
'Определяем путь к папке Мои документы
PathList = WshFldrs.item("MyDocuments") & "\"
'Создаем объект Folder для папки Мои документы
Set F = FSO.GetFolder(PathList)
'Создаем коллекцию файлов каталога Мои документы
Set Files = F.Files
s = "Файлы из каталога " & PathList & VbCrLf
'Цикл по всем файлам
For Each File In Files
'Добавляем строку с именем файла
s = s & File.Name & VbCrLf
Next
'Выводим полученные строки на экран
```

WScript.Echo s

Пример сценария для создания в каталоге «E:\Users» подкаталогов «My Folder»

```
'Объявляем переменные
Dim FSO, F, SubFolders

'Создаем объект FileSystemObject
Set FSO = WScript.CreateObject(''Scripting.FileSystemObject'')

'1. способ
FSO.CreateFolder("E:\Users\My Folder")

'2. способ
'Создаем объект Folder для каталога C:\Program Files
Set F = FSO.GetFolder(''E:\Users '')
'Создаем коллекцию подкаталогов каталога E:\Users
Set SubFolders = F.SubFolders
'Создаем каталог E:\Users\My Folder
SubFolders.Add ''My Folder''
```

Пример сценария, реализующего операции записи и чтения строк из текстового файла.

```
Dim FSO,F,TextStream,s  'Объявляем переменные
' Инициализируем константы
Const ForReading = 1, ForWriting = 2, TristateUseDefault = -2

' Создаем объект FileSystemObject
Set FSO=WScript.CreateObject(''Scripting.FileSystemObject'')
' Создаем в текущем каталоге файл test1.txt
FSO.CreateTextFile ''test1.txt''
' Создаем объект File для файла test1.txt
set F=FSO.GetFile(''test1.txt'')
' Создаем объект TextStream (файл открывается для записи)
Set TextStream=F.OpenAsTextStream(ForWriting, TristateUseDefault)
' Записываем в файл строку
TextStream.WriteLine ''Это первая строка''
' Закрываем файл
TextStream.Close
' Открываем файл для чтения
```

```
Set TextStream=F.OpenAsTextStream(ForReading, TristateUseDefault)
' Считываем строку из файла
s=TextStream.ReadLine
' Закрываем файл
TextStream.Close
' Отображаем строку на экране
WScript.Echo "Первая строка из файла test1.txt:" & vbCrLf & vbCrLf & s
```

Пример сценария, в котором на диске E создается файл TestFile.txt, который затем копируется на рабочий стол.

```
'Объявляем переменные
Dim FSO,F,WshShell,WshFldrs,PathCopy

'Создаем объект FileSystemObject
Set FSO = WScript.CreateObject("Scripting.FileSystemObject")
'Создаем файл
Set F = FSO.CreateTextFile("E:\TestFile.txt", true)
'Записываем в файл строку
F.WriteLine "Тестовый файл"
'Закрываем файл
F.Close

'Создаем объект WshShell
Set WshShell = WScript.CreateObject("Wscript.Shell")
'Создаем объект WshSpecialFolders
Set WshFldrs = WshShell.SpecialFolders
'Определяем путь к рабочему столу
PathCopy = WshFldrs.item("Desktop")+"\"
'Создаем объект File для файла C:\TestFile.txt
Set F = FSO.GetFile("E:\TestFile.txt")
'Копируем файл на рабочий стол
F.Copy PathCopy
```

Пример сценария, реализующего удаление файла E:\Users\TestFile.txt.

```
'Объявляем переменные
Dim FSO,F,FileName
'Создаем объект FileSystemObject
Set FSO = WScript.CreateObject("Scripting.FileSystemObject")
```

```
'Задаем имя файла
FileName="E:\Users\TestFile.txt"
FSO.DeleteFile FileName
WScript.Echo "Файл удален"
```

5. КОМАНДНЫЙ ИНТЕРФЕЙС ОС UNIX (LINUX)

5.1. Организация файловой системы

Файловая система ОС Линукс имеет иерархическую структуру каталогов, каждый из которых имеет свое предопределенное назначение. Можно выделить следующие основные типичные каталоги:

- /bin – исполняемые файлы общего назначения;
- /boot – содержит образ загружаемого ядра;
- /dev – файлы устройств;
- /etc – конфигурационные файлы общего пользования;
- /home – домашние каталоги пользователей, включая программы и файлы личных предпочтений;
- /lib – общесистемные библиотеки;
- /mnt – каталог монтирования внешних файловых систем;
- /root – домашний каталог суперпользователя;
- /sbin – программы системного администрирования;
- /usr – каталог пользовательских программ.

Для быстрого доступа к домашнему каталогу пользователя в среде существует две переменных окружения – **$HOME** или **~** (тильда).

Пример

cd $HOME или cd ~ переход в домашний каталог пользователя.

5.2. Основные команды Linux для работы с процессами

Рассмотрим основные команды Linux по управлению процессами.

Команда **top** – показывает список работающих в данный момент процессов с информацией о них. Список интерактивно формируется в реальном времени, для выхода клавиша **q**.

Команда **ps [**Опции**] [number]** – команда для вывода информации о процессах. Основные опции:

- a все терминальные процессы;
- e все процессы;
- f генерировать полный листинг;
- l генерировать листинг в длинном формате.

Команда **ps** без параметров выводит информацию только об активных процессах, запущенных с данного терминала, в том числе и фоновых.

Команда **nice [-приращение приоритета] команда[аргументы]** – команда изменения приоритета. Каждое запущенное приложение (процесс) имеет номер приоритета в диапазоне от 0 до 39, на основе которого ядро вычисляет фактический приоритет, используемый для планирования процесса. Значение 0 представляет наивысший приоритет, а 39 – самый низший.

Пример

nice -10 ls -l – увеличение приоритета процессу ls -l на 10.

Команда **kill [-sig] <идентификатор процесса>** – прекращение выполнения процесса. sig – номер сигнала: Sig – 15 означает программное (нормальное) завершение процесса, номер сигнала – 9 – уничтожение процесса. По умолчанию sig= -9. Вывести себя из системы можно командой kill -9 0. Пользователь с низким приоритетом может прервать процессы, связанные только с его терминалом.

Команда **free** – вывод информации о использовании памяти операционной системой.

Команда **tty** – информация о терминалах пользователя.

5.3. Команды работы с файловой системой

Для управления файловой системой имеются различные команды, реализующие операции по созданию, чтению, копированию, переименованию/перемещению, изменению и удалению файлов и каталогов. Рассмотрим основные из них.

Команда **echo [-n] [arg1] [arg2] [arg3]... –** на стандартный вывод строки символов, которая задана ей в качестве аргумента. При наличии флага *-n* символ перевода строки исключается.

Пример

$echo "Hello, world!"

Команда **pwd** – выдача имени текущего каталога.

Команда **cd [каталог]** – смена текущего каталога.

Команда **cat <имя файла>** – вывод содержимого файла на экран.

Примеры

cat > text.1 – создает новый файл с именем **text.1**, который можно заполнить символьными строками, вводя их с клавиатуры. Для окончания ввода – нажать ***Ctrl - d***.

cat text.1 > text.2 – пересылает содержимое файла **text.1** в файл **text.2**.

cat text.1 text.2 > text.3 – слияние файлов **text.1 text.2** в **text.3**.

Команда **mkdir [-m режим_доступа] [-p] каталог –** создание каталогов. Командой mkdir обрабатываются две опции:

- **-m режим_доступа** – задание параметров доступа синтаксис как у **chmod** (с режимом доступа по умолчанию **0777**);
- **-p** перед созданием нового каталога предварительно создаются все вышележащие каталоги.

Команда **ls [-опции] [имя]** – вывод содержимого каталога на экран. Если аргумент не указан, выдается содержимое текущего каталога. Основные аргументы команды:

- l – список включает всю информацию о файлах;
- a – в список включаются все файлы, в том числе и те, которые начинаются с точки;
- d – вывести имя самого каталога, но не содержимое;
- r – сортировка строк вывода;
- v – сортировка файлов по времени последнего доступа;
- c – использовать время создания файла при сортировке;
- g – то же, что -l, но с указанием имени группы пользователей;
- f – вывод содержимого всех указанных каталогов;
- R – рекурсивный вывод содержимого подкаталогов заданного каталога.

Пример

ls -l file.1 – чтение атрибутов файла **file.1**.

Команда **cp файл1 [файл2 ...] целевой_файл –** команда копирования файлов. Если целевой_файл является каталогом, то файл1, файл2, ..., копируются в него под своими именами. Только в этом случае можно указывать несколько исходных файлов.

Пример

cp file.1 file.2 – копирование файла **file.1** с переименованием.

Команда **mv [-f] файл1 [файл2 ...] целевой_файл** – команда перемещения (переименования) файлов. Если целевой_файл является каталогом, то файл1, файл2, ..., перемещаются в него под своими именами. Если была указана опция –f, действия выполняются без запросов подтверждения.

Пример

mv file.1 file.2 – переименование файла file.1 в file.2;

mv file.1 file.2 file.3 directory – перемещение файлов file.1, file.2, file.3 в указанную директорию.

Команда **rm** – команда служит для удаления указанных имен файлов из каталога. Синтаксис команды:

rm [-f] [-i] файл ...
rm -r [-f] [-i] каталог ... [файл ...]

Допускаются следующие опции:

-f – команда не выдает сообщений, когда удаляемый файл не существует, не запрашивает подтверждения;

-r – рекурсивное удаление всех каталогов и подкаталогов, перечисленных в списке аргументов. Сначала каталоги опустошаются, затем удаляются;

-i – запрос подтверждения перед удалением каждого файла. Опция -i отменяет действие опции -f.

Примеры

rm file.1 file.2 file.3 – удаление файлов file.1, file.2, file.3;

rm -ir dirname – запрос подтверждения на удаление всех содержащихся в каталоге файлов (для подкаталогов выполняются те же действия), а затем подтверждение на удаление самого каталога.

Команда **rmdir каталог ...** – команда удаляет пустые каталоги.

Команда **ln (link) –** создание ссылок. Один файл можно сделать принадлежащим нескольким каталогам. Для этого используется команда:

ln <имя файла 1> <имя файла 2> – создание ссылки на файл 1, с именем 2-го файла

ln -s <имя файла 1> <имя файла 2> – создание символической связи. Символическая связь является особым видом файла, в котором хранится имя файла, на который символическая связь ссылается. При просмотре или редактировании содержимого символической связи работа осуществляется с данными файла, на который эта связь ссылается.

5.4. Режимы доступа к файлам

В LINUX различаются 3 уровня доступа к файлам и каталогам: 1) доступ владельца файла; 2) доступ группы пользователей, к которой принадлежит владелец файла; 3) остальные пользователи. Для каждого уровня существуют свои байты атрибутов, значение которых

расшифровывается следующим образом: **r** – разрешение на чтение; **w** – разрешение на запись; **x** – разрешение на выполнение; **-** – отсутствие разрешения.

Первый символ байта атрибутов определяет тип файла и может интерпретироваться со следующими значениями: **-** – обычный файл; **d** – каталог; **l** – символическая связь.

В домашнем каталоге пользователь имеет полный доступ к файлам (**READ - r, WRITE - w, EXECUTE - x**). Атрибуты файла можно просмотреть командой **ls -l** и они представляются в следующем формате:

```
d rwx rwx rwx
|  |   |  |Доступ для остальных пользователей
|  |   |Доступ к файлу для членов группы
|  |Доступ к файлу владельца
|Тип файла (директория)
```

Пример

Командой **ls -l** получим листинг содержимого текущей директории **student**:

```
- rwx --- --- 2 student 100 Mar 10 10:30 file_1
- rwx --- r-- 1 adm 200 May 20 11:15 file_2
- rwx --- r-- 1 student 100 May 20 12:50 file_3
```

Атрибуты файла и доступ к нему можно изменить командой:

chmod <коды защиты> <имя файла>

Коды защиты могут быть заданы в числовом или символьном виде. Для символьного кода используются:

- знак плюс (+) – добавить права доступа;
- знак минус (-) – отменить права доступа;
- r, w, x – доступ на чтение, запись, выполнение;
- u, g, o – владельца, группы, остальных.

Коды защиты в числовом виде могут быть заданы в восьмеричной форме.

Пример

chmod g+rw,o+r file.1 – установка атрибутов чтения и записи для группы и чтения для всех остальных пользователей;

chmod o-w file.1 – отмена атрибута записи у остальных пользователей.

5.5. Создание командных файлов

Командный файл в Unix представляет собой обычный текстовый файл, содержащий набор команд Shell. Для того, чтобы командный интерпретатор воспринимал этот текстовый файл как командный, необходимо установить атрибут на исполнение.

Установку атрибута на исполнение можно осуществить командой **chmod**.

Пример

```
$ echo " ps -af " > commandfile
$ chmod +x commandfile
$ ./commandfile
```

6. BASH SHELL ОС UNIX (LINUX)

Язык Shell по своим возможностям приближается к высокоуровневым алгоритмическим языкам программирования. Текст процедуры набирается как текстовый файл и может быть вызван на исполнение одним из следующих способов.

```
$ chmod u+x shfile        # сделать файл исполняемым
$ shfile                  # запустить на выполнение
```

$ sh -c "shfile" или ***$ sh shfile*** *# с помощью* sh вызывается интерпретатор shell

6.1. Системные SHELL переменные

Процедуре при ее запуске могут быть переданы аргументы. В общем случае командная строка вызова процедуры имеет следующий вид:

$ имя_процедуры $1 $2 …$9

Параметр $0 соответствует имени самой процедуры. Сам интерпретатор shell автоматически присваивает значения следующим переменным (параметрам):

$? *Значение, возвращенное последней командой;*
$$ *номер процесса;*
$! *Номер фонового процесса;*
$# *число позиционных параметров, передаваемых в shell;*
$* *перечень параметров, как одна строка;*
$@ *перечень параметров, как совокупность слов;*
$- *флаги, передаваемые в shell.*

Пример

Вызов файла – *P1 par1 par2 par3*, имеющего вид:

echo $0 – имя файла
echo $? – код завершения
echo $$ – идентификатор последнего процесса
echo $! – идентификатор последнего фонового процесса
echo
echo $* – значения параметров, как строки
echo $@ – значения параметров, как слов
echo
set –au
echo $- – режимы работы интерпретатора

Выдаст на экран

P1 – имя файла
0 – код завершения
499 – идентификатор последнего процесса
98 – идентификатор последнего фонового процесса
par1 par2 par3 – значения параметров, как строки
par1 par2 par3 – значения параметров, как слов
au – режим работы интерпретатора

Количество передаваемых аргументов может быть увеличено путем «сдвига» их в командной строке влево на одну позицию с помощью команды **shift** без аргументов. После выполнения **shift** прежнее значение параметра $1 теряется, значение $1 приобретает значение $2, значение $2 – значение $3 и т. д.

Значения параметрам, передаваемым процедуре, можно присваивать и в процессе работы процедуры с помощью оператора **set**. Запуск **set** без параметров выводит список установленных системных переменных.

Пример

Вызов файла – *P1*, имеющего вид:

set a b c
echo первый=$1 второй=$2 третий=$3

выдаст на экран:

первый=a второй=b третий=c

Удаление переменных может быть выполнено с помощью команды:

$unset перем1 [перем2 ……..]

Для ввода строки текста с клавиатуры используется оператор:

read имя1 [имя2 имя3]

Если имен больше, чем полей в строке, то оставшиеся переменные будут инициализированы пустым значением. Если полей больше, чем имен переменных, то последней переменной будет присвоена подстрока введенной строки, содержащая все оставшиеся поля, включая разделители между ними.

Пример

#Текст процедуры:
echo "Введите значения текущих: гг мм ччвв"
read 1v 2v 3v
echo "год 1v"
echo "месяц 2v"
echo "сегодня 3v"

Кавычки используются для блокирования пробелов. Результат выполнения процедуры:

Введите значения текущих: гг мм ччвв
2005 Март 21 9:30 <Enter>
год 2005
месяц Март
сегодня 21 9:30

В Shell при написании операторов важное значение отводится кавычкам:

'…' – для блокирования специальных символов, которые могут быть интерпретированы как управляющие;

«…» – блокирование наиболее полного набора управляющих символов или указания того, что здесь будет обрабатываться не сам аргумент, а его значение;

`…` – (обратные кавычки) для указания того, что они обрамляют команду и здесь будет обрабатываться результат работы этой команды.

Пример

$echo `ls`	***$echo ``ls``***	***$ set `date`***
fil.1		***$ echo $3***
fil.2		***14:30:25***

6.2. Управление локальными переменными

Переменные, использующиеся в теле процедур, называются **локальными переменными**. Для определения значения переменной используется символ **$**.

Пример

$ z=123	***$ y=abc***	***$ var=/user/lab/ivanov***	***$ A=1 B=2***
$ echo $z	***$ echo "y"***	***$ cd $var***	***$ dat="$A + $B"***
123	***abc***	***$ pwd***	***$ echo $dat***
		/udd/lab/ivanov$	***1 + 2***

С переменными можно выполнять арифметические действия с использованием специального оператора:

expr – вычисление выражений.

Для ***арифметических операций*** используются операторы: **+** сложение; - вычитание; * умножение (обратная прямая скобка \ используется для отмены действия управляющих символов, здесь *); / деление нацело; % остаток от деления. Для **логических операций** используются следующие обозначения: = равно; != не равно; \< меньше; \<= меньше или равно; \> больше; \>= больше или равно.

Пример

```
# Текст процедуры:
a=2
a=`expr $a + 7`
b=`expr $a / 3`
c=`expr $a – 1 + $b`
d=`expr $c % 5`
e=`expr $d - $b`
echo $a $b $c $d $e
```

Результат работы процедуры:

9 3 11 1 -2

6.3. Экспортирование локальных переменных в среду SHELL

При выполнении процедуры ей можно передавать как позиционные параметры, так и ключевые – локальные переменные порожденного процесса. Для этого локальная переменная должна быть экспортирована (включена) в среду, в которой исполняется процедура, использующая эту переменную. Три формата команды экспортирования:

$export список имен локальных переменных

$export имя_лок_переменной=значение

$export (без параметров) – выводит перечень всех экспортированных локальных и переменных среды.

Пример

$color = red # переменная определена, но не экспортирована

$export count = 1 # переменная определена и экспортирована, т. е. потенциально доступна всем порождаемым процессам.

6.4. Команды проверки условий. Ветвление

Для проверки выполнения некоторого условия можно использовать команду:

test условие или просто ***[условие]***

Между скобками и содержащимся в них условием **обязательно** должны быть пробелы. Аргументами этой команды могут быть имена файлов, числовые или нечисловые строки (цепочки символов). Командой вырабатывается код завершения (код возврата), соответствующий закодированному в команде **test** условию. Если закодированное параметрами условие выполняется, то вырабатывается логический результат – **true**, если нет – **false**.

Код возврата может обрабатываться как следующей за **test** командой, так и специальной конструкцией языка: **if-then-else-fi**. В shell используются условия различных «типов».

Таблица 6.1

Условия проверки файлов

Тип	Описание
-f file	файл "file" является обычным файлом
-d file	файл "file" – каталог
-c file	файл "file" – специальный файл
-r file	имеется разрешение на чтение файла "file"
-w file	имеется разрешение на запись в файл "file"
-x file	имеется разрешение на исполнение файла "file"
-s file	файл "file" не пустой

Пример

[-f myfile] # проверка является ли myfile файлом
echo $? # вывод кода возврата
0

Таблица 6.2

Условия проверки строк

Тип	Описание
str1 = str2	строки "str1" и "str2" совпадают
str1 != str2	строки "str1" и "str2" не совпадают
-n str1	строка "str1" существует (не пустая)
-z str1	строка "str1" не существует (пустая)

Пример

x="who is who"
["who is who" = "$x"] # сравнение строк
echo $?
0

Таблица 6.3

Условия сравнения целых чисел

Тип	Описание
x -eq y	"x" равно "y"
x -ne y	"x" неравно "y"
x -gt y	"x" больше "y"
x -ge y	"x" больше или равно "y"
x -lt y	"x" меньше "y"
x -le y	"x" меньше или равно "y"

Пример

$x=5
$[$x -lt 7]
$echo $?
0

Таблица 6.4

Логические операции

Тип	Описание
!	(not) инвертирует значение кода завершения
-o	(or) соответствует логическому "ИЛИ"
-a	(and) соответствует логическому "И"

В общем случае оператор "if" имеет структуру (команды в квадратных скобках являются не обязательными):

if условие
then Список_команд
[elif условие
then Список_команд]
[else Список_команд]
fi

Список_команд представляет собой команду или несколько команд, либо фрагмент shell-процедуры. Если команды записаны на одной строке, то они разделяются точкой с запятой. Для задания пустого списка команд следует использовать специальный оператор **:** (двоеточие). В качестве условия могут использоваться списки любых команд. Однако чаще других используется команда **test**. В операторе **if** так же допускается две формы записи этой команды:

if test аргументы или if [аргументы]

Каждый оператор **if** произвольного уровня вложенности обязательно должен завершаться словом **fi**.

Пример

Создать файл, сравнивающий передаваемый ему параметр с некоторым набором символов (паролем).

if test 'param' = "$1" – сравниваются строки символов
then echo Y
else echo N
fi

Пример

Организовать ветвление вычислительного процесса в зависимости от значения переменной X (<10, >10, = 10).

```
If [ $X -lt 10 ]
then  echo X is less 10
else
 if [ $X -gt 10 ]
 then  echo X is greatr 10
 else
 echo X is equal to 10
 fi
fi
```

Для задания множественного ветвления в зависимости от значений аргументов используется команда **case**. Формат:

```
case <string> in
s1) <list1>;;
s2) <list2>;;
.
sn) <listn>;;
*) <list>
esac
```

Здесь **list1**, **list2**, ..., **listn** – список команд. Производится сравнение шаблона **string** с шаблонами **s1**, **s2**,..., **sn**. При совпадении выполняется список команд, стоящий между текущим шаблоном и соответствующими знаками **;;**. Обычно последняя строка выбора имеет шаблон "*", что означает "любое значение".

Пример

Определить, к какой возрастной группе относится введенный возраст.

```
echo -n ' Введите ваш возраст '
read age
case $age in
 test $age -le 20) echo 'Группа 1' ;;
 test $age -le 40) echo 'Группа 2' ;;
 test $age -le 70) echo 'Группа 3' ;;
 100) echo 'Группа 100' ;;
 *) echo ''Группа 4'
esac
```

6.5. Команды организации циклов

Циклы обеспечивают многократное выполнение отдельных участков процедуры до достижения заданных условий. Оператор цикла **while**, синтаксис:

```
while условие
   do список_команд
done
```

Если условие истинно, выполняется список_команд, а если ложно, выполнение цикла завершается.

Пример

Выполнить проверку наличия аргументов при обращении к данной процедуре и в случае их наличия вывести на экран.

Текст процедуры, которой присвоено имя P1:

```
if $# –eq 0
 then echo “No param”
 else echo “Param:’;
 while test ‘$1’
 do
 echo “$1”
 shift
 done
fi
```

Результат работы процедуры:

```
$P1
No param
$P1 abc df egh
Param:
abc
df
egh
$
```

Пример

Вывести на экран слово строки (поле), номер которого (переменная N) указан в параметре при обращении к процедуре (P2). Процедура запрашивает ввод строки с клавиатуры. Номер слова вводится как аргумент процедуры.

Текст процедуры P2:

```
i=1 # счетчик номеров слов в строке, формируется при каждом выполнении цикла
N=$1 #значение первого параметра
echo "Введи строку: "
read a
set $a
while test $i -lt $N
 do
 i=`expr $i + 1` # формирование номера следующего слова
 shift
 done
echo "$N поле строки: \"$1\""
```

Пример работы процедуры P3:

```
$P3 2
Введи строку: aa bb cc dd
2 поле строки: "bb"
$
```

Оператор цикла **until** выполняется, пока условие ложно. Синтаксис:

```
until условие
  do список_команд
done
```

Пример

Выполнить цикл заданное число раз.

```
X = 1 # счетчик числа циклов
until test $X -gt 10 # задано число циклов = 10
do
 echo X is $X
 X = `expr $X + 1`
done
```

```
Оператор цикла for, синтаксис:
for имя_ переменной [in список_значений]
  do список_команд
done
```

Переменная, с указанным в команде именем, заводится автоматически. Переменной присваивается значение очередного слова из списка_значений и для этого значения выполняется список_команд. Количество итераций равно количеству значений в списке, разделенных пробелами (т. е. циклы выполняются, пока список не будет исчерпан).

Пример

Вывести список имен файлов текущего каталога.

```
list =`ls`  # создаем список файлов
for val in $list
do
 echo "$val"
done
echo end
```

Пример

Скопировать все обычные файлы из текущего каталога в каталог, который задается в качестве аргумента. Процедура проверяет так же наличие каталога-адресата и сообщает количество скопированных файлов.

```
m=0 # переменная для счетчика скопированных файлов
if [ -d $HOME/$1 ]
 then echo "Каталог $1 существует"
 else
 mkdir $HOME/$1 .
 echo "Каталог $1 создан"
fi
 for file in *
 do
 if [ -f "$file" ]
 then cp "$file" $HOME/$1;
 m=`expr $m + 1`
 fi
 done
echo "Число скопированных файлов: $m"
```

Здесь символ * – имеет смысл <список имен файлов текущего каталога>

Пример

Вывести на экран имена файлов из текущего каталога, число символов в имени которых не превышает заданного параметром числа.

```
if [ "$1" = "" ]
then
 exit
fi
for nam in *
do
 size = `expr $nam : '.*'`
 if [ "$size" –le "$1" ]
 then echo "Длина имени $nam $size символа"
 fi
done
```

Результаты работы процедуры:

```
$ PROC 2
 Длина имени f1 2 символа
 Длина имени f2 2 символа
$PROC 3
 Длина имени out 3 символа
 Длина имени f1 2 символа
 Длина имени f2 2 символа
$
```

Пример

Вывести на экран из указанного параметром подкаталога имена файлов с указанием их типа.

```
cd $1
for fil in *
do
 If [ -d $fil]
 then echo "$fil – catalog"
 else echo "$fil – file"
 fi
done
```

7. КОМАНДЫ УПРАВЛЕНИЯ И ТЕСТИРОВАНИЯ СЕТЕВЫХ РЕСУРСОВ

Интерфейсом, с точки зрения ОС, является устройство, через которое система получает и передает IP-пакеты. Роль интерфейса локальной сети может выполнять одно (или несколько) из следующих устройств: Ethernet-карта, ISDN-адаптер или модем, подключенный к последовательному порту. Каждое устройство (не весь компьютер!) имеет свой IP-адрес. Большинство команд управления сетевыми ресурсами работают с конкретным интерфейсом.

7.1. Команды ОС Windows

NETSTAT – показывает активные подключения через стек протоколов TCP/IP, порты, прослушиваемые компьютером, статистики Ethernet, таблицы маршрутизации IP, статистики IPv4 (для протоколов IP, ICMP, TCP и UDP) и IPv6 (для протоколов IPv6, ICMPv6, TCP через IPv6 и UDP через IPv6). Запущенная без параметров команда NETSTAT отображает подключения TCP.

Формат команды:

NETSTAT [-a] [-e] [-n] [-o] [-p *protocol*] [-r] [-s] [*interval*]

где

-a – вывод всех активных подключений TCP и прослушиваемых компьютером портов TCP и UDP;

-e – вывод статистики Ethernet, например, количества отправленных и принятых байтов и пакетов. Этот параметр может комбинироваться с ключом -s;

-n – вывод активных подключений TCP с отображением адресов и номеров портов в числовом формате без попыток определения имен;

-o – вывод активных подключений TCP и включение кода процесса (PID) для каждого подключения. Код процесса позволяет найти приложение на вкладке Процессы диспетчера задач Windows. Этот параметр может комбинироваться с ключами -a, -n и -p;

-p *protocol* – вывод подключений для протокола, указанного параметром протокол. В этом случае параметр *protocol* может принимать значения tcp, udp, tcpv6 или udpv6. Если данный параметр используется с ключом -s для вывода статистики по протоколу,

параметр *protocol* может иметь значение tcp, udp, icmp, ip, tcpv6, udpv6, icmpv6 или ipv6;

-s – вывод статистики по протоколу. По умолчанию выводится статистика для протоколов TCP, UDP, ICMP и IP. Если установлен протокол IPv6 для Windows XP, отображается статистика для протоколов TCP через IPv6, UDP через IPv6, ICMPv6 и IPv6. Параметр -p может использоваться для указания набора протоколов;

-r – вывод содержимого таблицы маршрутизации IP. Эта команда эквивалентна команде route print;

interval – обновление выбранных данных с интервалом, определенным параметром интервал (в секундах). Нажатие клавиш CTRL+C останавливает обновление. Если этот параметр пропущен, NETSTAT выводит выбранные данные только один раз.

IPCONFIG – показывает текущие сетевые настройки компьютера.

Формат команды:

IPCONFIG [/all | /release [adapter] | /renew [adapter]]

где

/all – показывает такие основную и дополнительную информацию, как сроки окончания аренды и службы разрешения имен;

/release – выдает IP-адрес указанному адаптеру, если адаптер использовал DHCP;

/renew – обновляет IP-адрес для указанного адаптера, если адаптер использовал DHCP.

При использовании без аргументов IPCONFIG представляет только основные настройки TCP/IP, включая IP-адрес, маску подсети и шлюз по умолчанию для каждой карты сетевого адаптера.

PING – команда, предназначенная для тестирования соединения между двумя сетевыми устройствами. В результате команда возвращает время прохождения сигнала.

Формат команды:

PING [-t] [-a] [-n] [-l] [-f] [-I TTL] [-v TOS] [-r] [-s] [-j список узлов]

[-k список узлов] [-w] список адресатов,

где

-t – поддерживает пингование, пока не будет остановлен нажатием клавиш CTRL+C;

-n – посылает эхо-сигнал определенное (указанное) количество раз и прекращает тестирование;

-l – посылает пакет с указанным количеством битов;

-f – устанавливает флаг Don't Fragment (не фрагментировать). Это значит, что пакеты не будут разбиваться на части сетевыми устройствами;

-w – устанавливает время простоя (мс). Время простоя по умолчанию равно 750 мс.

ARP – команда, позволяющая просматривать и редактировать таблицу ARP, которая формируется соответствующим протоколом из стека протоколов TCP/IP [3], [16].

Протокол разрешения адресов (Address Resolution Protocol, ARP) позволяет компьютерам создавать соединения на физическом уровне. Межсетевое взаимодействие устройств осуществляется на канальном уровне протоколов, где для передачи информации используются MAC-адреса сетевых плат. Поэтому адреса компьютеров сетевого уровня всегда конвертируются в MAC-адреса. Когда одна рабочая станция пытается установить связь с другой, она должна транслировать сигнал в соответствии с протоколом ARP, чтобы выяснить MAC-адрес. После того, как Windows XP Professional компьютер определит MAC-адрес, он использует его для установки связи с устройством. Эта конверсия IP в MAC хранится в ARP-таблице компьютера. Этот инструмент полезен при решении проблем, связанных с разрешениями имен.

Формат команды:

ARP -s inet_addr eth_addr [if_addr]
ARP -d inet_addr [if_addr]
ARP -a [inet_addr] [-N if_addr]

где

-s – добавляет IP-адрес (inet_addr) или Ethernet MAC адрес (eth_addr) в таблицу ARP. IP-адрес имеет стандартный четырехоктетный формат, в то время как Ethernet-адрес записывается шестью шестнадцатеричными значениями, разделенными тире;

-d – удаляет указанный IP-адрес из таблицы;

-a – выводит на экран текущую ARP-таблицу;

if_addr – указывает IP-адрес, отличный от данного по умолчанию.

TRACERT – команда, предназначенная для контроля маршрута перемещения пакета между двумя сетевыми устройствами. Она работает посредством передачи пакета со значением времени жизни (TTL), равным 1. Обычно маршрутизаторы сокращают значение TTL на 1 и затем отправляют пакет дальше по пути следования. Если маршрутизатор получает TTL со значением 0, то он возвращает пакет отправителю как просроченный. Команда TRACERT выполняет это действие для первого маршрутизатора на пути следования пакета, добавляет 1 к TTL и затем отправляет новый пакет. Следующий пакет доходит до второго маршрутизатора и становится просроченным. Этот маршрутизатор возвращает пакет вместе с информацией о самом себе. Процесс повторяется, пока пакет не дойдет до нужного устройства, или пока количество переходов не достигнет максимального значения.

Формат команды:

TRACERT [-d] [-h maximum_hops] [-j host-list] [-w timeout] имя конечного устройства,
где

-d – препятствует разрешению адреса именам хостов;

-h maximum_hops – устанавливает верхнюю границу общего числа переходов, необходимых для нахождения нужной рабочей станции;

-j host-list – устанавливает свободный начальный маршрутизатор для всего списка хостов;

-w timeout – устанавливает время простоя (мс) для каждого перехода.

PATHPING – команда, представляющая собой комбинацию команд PING и TRACERT. Эта команда последовательно посылает информационные пакеты на каждый маршрутизатор по пути к месту назначения. Затем рассчитываются результаты на основании пакетов, возвращенных каждым маршрутизатором. Так как PATHPING показывает степень потери пакетов в любом маршрутизаторе или соединении, администратор может определить, какие именно маршрутизаторы и соединения вызывают проблемы в работе сети.

Формат команды:

PATHPING [-n] [-h maximum_hops] [-g host-list] [-p period] [-q num_queries] [-w timeout] [-T] [-R] target_name

где

-n – не разрешает присваивать адреса именам хостов;

-h maximum_hops – указывает максимальное количество изменений маршрута, необходимое для нахождения конечного пункта. Настройка по умолчанию предусматривает 30 переходов;

-p period – указывает время (мс) между двумя передачами пинг-сигнала. По умолчанию равно 250 мс;

-q num_queries – указывает количество запросов, посланных на каждый компьютер во время прохождения маршрута. Значение по умолчанию – 100;

-w timeout – указывает время (мс), отводимое на ожидание ответа. По умолчанию – 3000 мс (или 3 с).

7.2. Команды ОС Unix (Linux)

NETSTAT – команда, отображающая содержимое различных структур данных, связанных с сетью, в различных форматах в зависимости от указанных опций. Существует три формы использования команды NETSTAT. Первая форма команды показывает список активных сокетов (sockets) для каждого протокола. Вторая форма выбирает одну из нескольких других сетевых структур данных. Третья форма показывает динамическую статистику пересылки пакетов по сконфигурированным сетевым интерфейсам; аргумент интервал задает, сколько секунд собирается информация между последовательными показами.

Формат команды:

netstat [-Aan] [-f <address>] [-I <interface>] [-p <protocol>] [<system>] [core]

netstat [-n] [-s] [-i | -r] [-f <addresses>] [-I <interface>] [-p <protocol>] [<system>] [core]

netstat [-n] [-I <interface>] interval [<system>] [core]

где

a – показывать состояние всех сокетов; обычно сокеты, используемые серверными процессами, не показываются;

-A – показывать адреса любых управляющих блоков протокола, связанных с сокетами; используется для отладки;

-i – показывать состояние автоматически сконфигурированных (auto-configured) интерфейсов. Интерфейсы, статически сконфигури-

рованные в системе, но не найденные во время загрузки, не показываются;

-n – показывать сетевые адреса как числа. Netstat обычно показывает адреса как символы. Эту опцию можно использовать с любым форматом показа;

-r – показать таблицы маршрутизации. При использовании с опцией -s, показывает статистику маршрутизации;

-s – показать статистическую информацию по протоколам. При использовании с опцией -r, показывает статистику маршрутизации;

-f <addresses> – ограничить показ статистики или адресов управляющих блоков только указанным семейством адресов в <addresses>, в качестве которого можно указывать: inet – для семейства адресов AF_INET, или unix – для семейства адресов AF_UNIX;

-I <interface> – выделить информацию об указанном интерфейсе в отдельный столбец; по умолчанию (для третьей формы команды) используется интерфейс с наибольшим объемом переданной информации с момента последней перезагрузки системы. В качестве интерфейса можно указывать любой из интерфейсов, перечисленных в файле конфигурации системы, например, emd1 или lo0;

-p <protocol> – ограничить показ статистики или адресов управляющих блоков только протоколом с указанным именем_протокола, например, tcp.

Примеры

1) >netstat -Aan inet

Active Internet connections (servers and established)

Proto Recv-Q Send-Q Local Address Foreign Address State

tcp 0 0 0.0.0.0:111 0.0.0.0:* LISTEN

tcp 0 0 0.0.0.0:22 0.0.0.0:* LISTEN

tcp 0 0 192.168.102.125:22 192.168.102.1:33208 ESTABLISHED

udp 0 0 0.0.0.0:11 0.0.0.0:*

Здесь видно, что на компьютере зарегистрировано два TCP-обработчика (на портах 111 и 22), один UDP-обработчик на 11-м порту (понятие Listener, т. е. обработчик соединения для UDP не имеет смысла), а также установлено одно соединение с компьютера 192.168.102.1, исходящий порт 33208, к 22-у порту (это порт службы Secure Shell, предоставляющей удаленный терминальный доступ);

2) >netstat -nr

Kernel IP routing table

Destination	Gateway	Genmask	Flags	MSS	Window	irtt	Iface
10.72.128.101	0.0.0.0	255.255.255.255	UH	0	0	0	eth0
10.72.128.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0
127.0.0.0	0.0.0.0	255.0.0.0	U	0	0	0	lo
0.0.0.0	10.72.128.254	0.0.0.0	UG	0	0	0	eth0

При вызове netstat с параметром -r, он показывает таблицу маршрутизации ядра, подготовленную с помощью команды route. Опция -n заставляет netstat печатать IP-адреса вместо имен хостов и сетей;

3) >netstat -i

Kernel Interface table

Iface	MTU	Met	RX-OK	RX-ERR	RX-DRP	RX-OVR	TX-OK	TX-ERR	TX-DRP	TX-OVR	Flags
lo	0	0	3185	0	0	0	3185	0	0	0	BLRU
eth0	1500	0	972555	17	20	120	555735	111	0	0	BLRU

Когда netstat вызывается с параметром -i, он показывает статистику для сетевых интерфейсов. Если, кроме того, дается опция -a, он будет печатать все интерфейсы, представленные в ядре, а не только те, которые были отконфигурированы в настоящее время. Поля MTU и Met показывают текущий MTU и метрическое значение для этого интерфейса. Колонки RX и TX показывают сколько пакетов были получены или переданы без ошибок (RX-OK /TX-OK), повредились (RX-ERR /TX-ERR), сколько было потеряно (RX-DRP/TX-DRP) и сколько было потеряно из-за переполнения (RX-OVR/ TX-OVR). Последняя колонка показывает флаги, установленные для этого интерфейса. Здесь используется односимвольная версия флагов, которые печатает ifconfig:

B – установлен широковещательный адрес;

L – этот интерфейс задает устройство loopback;

M – интерфейс получает все пакеты (режим promiscuous);

O – ARP выключен для этого интерфейса;

P – это соединение point-to-point;

R – интерфейс работает;

U – интерфейс активен.

IFCONFIG (“interface configuration”) – команда, предназначенная для настройки сетевых интерфейсов. Настройка интерфейса заключается в присвоении IP-адресов сетевому устройству и установке нужных значений для других параметров сетевого подключения.

Формат команды:

ifconfig [-h] [-a] [<interface>] [<address>] [-broadcast <address>] [-netmask <address>] [-up|-down]

где

-a – показывает информацию о всех интерфейсах. Если данная опция отсутствует, выводится информация только об активных интерфейсах;

interface – конфигурировать или показать информацию только о заданном интерфейсе;

address – IP-адрес, присваиваемый интерфейсу;

-broadcast <address> – широковещательный адрес, присваиваемый интерфейсу, address – широковещательный адрес;

-netmask <address> – маска подсети, используемая совместно с адресом, address – маска. Если маска не задана явно, то маска принимается, равной стандартным значения для стандартных классов подсетей A, B и C [16];

-up – активирует интерфейс. При активизации интерфейса для него автоматически добавляется соответствующий маршрут в таблице маршрутизации;

-down – деактивирует интерфейс. При деактивации интерфейса соответствующий маршрут автоматически удаляется из таблицы маршрутизации.

Если ifconfig вызывается без параметров, то команда выводит на экран данные о состоянии всех активных интерфейсов.

Примеры

1) >ifconfig

```
eth0    Link encap:Ethernet  HWaddr 0:0:0:0:CF:0
inet        addr:192.168.120.1        Bcast:192.168.120.255
Mask:255.255.255.0
          UP
          RX packets:23 errors:0 dropped:0
          TX packets:23 errors:0 dropped:0
          RX bytes:0 TX bytes:0
```

где

HWaddr – уникальный шестибайтовый адрес интерфейса, аналогичный MAC-адресу в Ethernet сетях, назначается автоматически;

2) >ifconfig eth0 192.168.120.1 -up

>ifconfig

```
eth0    Link encap:Ethernet  HWaddr 0:0:0:0:CF:0
 inet addr:192.168.120.1 Bcast:192.168.120.255 Mask:255.255.255.0
       UP
       RX packets:0 errors:0 dropped:0
       TX packets:0 errors:0 dropped:0
       RX bytes:0 TX bytes:0
```

3) > ifconfig eth0 192.168.0.15 -netmask 255.255.255.0 -up

```
>ifconfig
eth0    Link encap:Ethernet  HWaddr 0:0:0:0:CF:0
inet addr:192.168.0.15  Bcast:192.168.0.255  Mask:255.255.255.0
       UP
       RX packets:0 errors:0 dropped:0
       TX packets:0 errors:0 dropped:0
       RX bytes:0 TX bytes:0
```

Как правило используются три основных интерфеса:

lo – локальный интерфейс;

ethN – интерфейс для платы Ethetnet, где N – число – номер интерфейса начиная от 0;

PPP (SLIP) – интерфейс для последовательного порта.

Локальный интерфейс lo используется для связи программ IP-клиентов с IP-серверами, запущенными на том же компьютере, поэтому в любом случае его необходимо настроить.

Для настройки локального интерфейса необходимо выполнить команду:

```
ifconfig lo 127.0.0.1
```

В некоторых случаях необходимо бывает изменить адрес прерывания, используемого сетевой картой, порта ввода-вывода или сетевого протокола канального уровня. Это можно сделать, выполнив следующую команду:

ifconfig eth0 irq 5 io_addr 220 media 10baseT

Замечание. Не все устройства (платы) поддерживают динамическое изменение этих параметров.

PING – команда, проверяющая наличие сетевого соединения до указанного устройства. PING использует ICMP протокол чтобы проверить достижимость интерфейса удаленного узла. PING посылает удаленному узлу ICMP ECHO_REQUEST и ожидает в течение определенного промежутка времени ICMP ECHO_RESPONSE. В случае получения ответа выводит данные о прохождении ICMP-пакета по сети.

Формат команды:

ping [-h] [-i <interval>] [-t <ttl>] <destination>

где

-i <interval> – задает частоту ICMP-запросов, interval – интервал между запросами в секундах (по умолчанию отсылается один пакет в секунду);

-t <ttl> – задает значение атрибута Time to Live в генерируемых IP-пакетах, ttl — целое число 0-255 (по умолчанию TTL равно 64);

destination– IP-адрес исследуемого узла.

Пример

```
>ping 192.168.0.1
  PING 192.168.0.1
  64 bytes from 192.168.0.1: icmp_seq=0 ttl=62 time=135 ms
  64 bytes from 192.168.0.1: icmp_seq=1 ttl=62 time=155 ms
  64 bytes from 192.168.0.1: icmp_seq=2 ttl=62 time=55 ms
  64 bytes from 192.168.0.1: icmp_seq=3 ttl=62 time=48 ms
  64 bytes from 192.168.120.1: icmp_seq=4 ttl=62 time=97 ms
```

Это означает, что сетевое соединение работает. Для того чтобы прервать тестирование сети, нажмите комбинацию клавиш <Ctrl>+<C>.

PING выводит результат исследования удаленного узла в следующем формате: 64 bytes from 192.168.0.1 – размер полученного ответа и адрес источника ответа. icmp_seq=0 – номер пакета. Каждый запрос содержит свой номер, как правило формируется инкрементно. PING выводит номер пакета из каждого полученного ответа. ttl=62 – значение TTL из полученного ответа. time=55 ms – время прохождения пакетом полного маршрута (туда и обратно, round-trip time) в миллисекундах.

TRACEROUT – команда, позволяющая приблизительно определять маршрут следования пакета, в результате отображает список абонентов, через которые проходит пакет по пути к адресату, и потраченное на это время.

Замечание. Однако список этот приблизительный. Дело в том, что первому пакету (точнее, первым трем, так как по умолчанию traceroute шлет пакеты по три) в специальное поле TTL (Time To Live, время жизни) выставляется значение "1". Каждый маршрутизатор должен уменьшать это значение на 1, и если оно обнулилось, передавать отправителю ICMP-пакет о том, что время жизни закончилось, а

адресат так и не найден. Так что на первую серию пакетов отреагирует первый же маршрутизатор, и traceroute выдаст первую строку маршрута. Второй пакет посылается с TTL=2 и, если за две пересылки адресат не достигнут, об этом рапортует второй маршрутизатор. Процесс продолжается до тех пор, пока очередной пакет не «доживет» до места назначения. Строго говоря, неизвестно, каким маршрутом шла очередная группа пакетов, потому что с тех пор, как посылалась предыдущая группа, какой-нибудь из промежуточных маршрутизаторов мог передумать и послать новые пакеты другим путем.

Пример

>traceroute www.ya.ru -n
traceroute to www.ya..ru (214.87.0.50), 30 hops max, 38 byte packets
1 192.168.0.100 0.223 ms 0.089 ms 0.105 ms
2 83.237.29.1 25.599 ms 21.390 ms 21.812 ms
3 195.34.53.53 24.111 ms 21.213 ms 25.778 ms
4 195.34.53.53 23.614 ms 33.172 ms 22.238 ms
5 195.34.53.10 43.552 ms 48.731 ms 44.402 ms
6 195.34.53.81 26.805 ms 21.307 ms 22.138 ms
7 213.248.67.93 41.737 ms 41.565 ms 42.265 ms
8 213.248.66.9 50.239 ms 47.081 ms 64.781 ms
9 213.248.65.42 99.002 ms 81.968 ms 62.771 ms
10 213.248.78.170 62.768 ms 63.751 ms 78.959 ms
11 214.87.0.66 101.865 ms 88.289 ms 66.340 ms
12 214.87.0.50 70.881 ms 67.340 ms 63.791 ms

ARP – команда, показывающая ARP-таблицу устройства. Кроме того, опция -r позволяет сформировать запрос для определения MAC-адреса по явно заданному IP-адресу.

Формат команды:

arp [-r <IP-address> <interface>]

где

-r <IP-address> <interface>. Прежде чем вывести ARP-таблицу предпринимает попытку найти MAC-адрес по явно заданному IP-адресу, <IP-address> – IP-адрес, для которого определяется MAC-адрес, <interface> – имя интерфейса, в сети подсоединенной к которому будет происходить поиск.

Примеры

1) >arp
   Address HWaddress iface

10.0.0.10 0:0:0:0:BC:0 eth0
10.0.0.11 0:0:0:0:1F:2 eth0

Если arp вызывается без параметров, то команда выводит на экран ARP-таблицу;

2)>arp -r 192.168.120.12 eth1
Address HWaddress iface
10.0.0.10 0:0:0:0:BC:0 eth0
10.0.0.11 0:0:0:0:1F:2 eth0
192.168.120.12 0:0:0:0:12:1 eth1

MACTABLE – команда, показывающая таблицу MAC-адресов коммутаторов второго уровня.

Формат команды:

mactable

Пример

>mactable
MACAddress port
0:0:0:0:B3:0 0
0:0:0:0:2F:2 0
0:0:0:0:03:0 3

где port – номер порта на коммутаторе.

ROUTE – команда, позволяющая управлять таблицей маршрутизации устройств поддерживающих протокол IP4.

Формат команды:

route [{-add|-del} <target> [-netmask <address>] [-gw <address>] [-metric <M>] [-dev <If>]]

где

<target> – адрес назначения. Назначением может быть подсеть или отдельный узел в зависимости от значения маски подсети. Если маска равна 255.255.255.255 или отсутствует совсем, то назначением будет узел, иначе назначением будет сеть;

-add– добавляет новый маршрут в таблицу маршрутизации;

-del – удаляет маршрут из таблицы маршрутизации;

-dev <If> – принудительно присоединяет маршрут к определенному интерфейсу, <If> – имя интерфейса;

-gw <address> – направляет пакеты по этому маршруту через заданный шлюз, <address> – адрес шлюза;

-netmask <address> – маска подсети, используемая совместно с адресом назначения при добавлении маршрута, <address> – маска, если маска не задана явно подразумевается стандартная маска, соответствующая таблице IP-адресов;

-metric <M> – метрика, используемая в данном маршруте, M – целое число, большее или равное нулю.

Примеры

1) >route

```
IP routing table
Destination     Gateway      Netmask         Flags  Metric  Iface
10.0.0.0        *            255.0.0.0       U      1       eth0
11.0.0.0        10.0.0.10    255.0.0.0       UG     1       eth0
192.168. 0.1    10.0.0.10    255.255.255.255 UGH    1       eth0
```

Если route вызывается без параметров, то команда выводит на экран таблицу маршрутизации. Если маршрут не использует шлюз, вместо адреса шлюза выводится *. Поле «Flags» может содержать значение: U – маршрут активен, G – маршрут использует шлюз, H – назначением является узел.

2) >route -add 192.168. 0.0 -netmask 255.255.255.0 -dev eth0

>route

```
IP routing table
Destination     Gateway      Netmask         Flags  Metric  Iface
192.168. 0.0    *            255.255.255.0   U      1       eth0
```

3) >route -add 192.168.1.0 -gw 192.168.0.10

>route

```
IP routing table
Destination     Gateway      Netmask         Flags  Metric  Iface
192.168. 0.0    *            255.255.255.0   U      1       eth0
192.168.1.0     192.168.0.10 255.255.255.0   UGH    1       eth0
```

ЛИТЕРАТУРА

1. Таненбаум, Э. Современные операционные системы / Э. Таненбаум. – Санкт-Петербург : Питер, 2002.

2. Столингс, В. Операционные системы / В. Столингс. – 3-е изд. – Москва : Вильямс, 2002.

3. Олифер, В. Г. Сетевые операционные системы : учебник / В. Г. Олифер, Н. А. Олифер. – Санкт-Петербург : Питер, 2001.

4. Гордеев, А. В. Операционные системы / А. В. Гордеев. – Санкт-Петербург : Питер, 2006.

5. Щупак, Ю. А. Win API. Эффективная разработка приложений / Ю. А. Щупак. – Санкт-Петербург : Питер, 2006.

6. Галатенко, В. Программирование в стандарте POSIX : курс лекций / В. Галатенко. – 2006.

7. Microsoft Windows 2000: Server и Professional. Русские версии / Андреев А. Г. [и др.] ; под общ. ред. А. Н. Чекмарева и Д. Б. Вишнякова. – Санкт-Петербург : БХВ-Петербург, 2001.

8. Петерсен, Р. Linux: руководство по операционной системе / Р. Петерсен. – Киев : BHV, 1998.

9. Таненбаум, Э. Операционные системы. Разработка и реализация / Э. Таненбаум, А. Вудхалл. – Санкт-Петербург : Питер, 2007.

10. Бройдо, В. Л. Архитектура ЭВМ и систем / В. Л. Бройдо, О. П. Ильина. – Санкт-Петербург : Питер.

11. Крелл, М. Linux. Администрирование сетей TCP/IP / М. Крелл, С. Манн. – Москва : Вильямс, 2003.

12. Стахнов, А. Сетевое администрирование Linux / А. Стахнов. – Санкт-Петербург : Питер-пресс, 2004.

13. Петерсен, Р. Linux: руководство по операционной системе / Р. Петерсен. – Киев : BHV, 1998.

14. Администрирование сети на основе Windows 2000. Учебный курс MCSE. Сертификационный экзамен 70-216. – Санкт-Петербург : БХВ-Петербург, 2004.

15. Информатика: Базовый курс / С. В. Симонович [и др.]. – Санкт-Петербург : Питер, 2001.

16. Курочка, К. С. Компьютерные сети : практ. пособие курсов «Сетевые технологии», «Компьютерные информационные технологии», «Информатика» для студентов днев. и заоч. отделений. – Гомель, 2005.

Содержание

Учебное электронное издание комбинированного распространения

Учебное издание

Курочка Константин Сергеевич
Литвинов Дмитрий Александрович

ОПЕРАЦИОННЫЕ СИСТЕМЫ

Пособие
по одноименному курсу
для студентов специальности 1-40 01 02
«Информационные системы и технологии (по направлениям)»

Электронный аналог печатного издания

Редактор *Н. И. Жукова*
Компьютерная верстка *Е. В. Темная*

Подписано в печать 06.10.09.
Формат 60x84/16. Бумага офсетная. Гарнитура «Таймс».
Ризография. Усл. печ. л. 3,95. Уч.-изд. л. 3,86.
Изд. № 237.
E-mail: ic@gstu.gomel.by
http://www.gstu.gomel.by

Издатель и полиграфическое исполнение:
Издательский центр учреждения образования
«Гомельский государственный технический университет
имени П. О. Сухого».
ЛИ № 02330/0549424 от 08.04.2009 г.
246746, г. Гомель, пр. Октября, 48.