

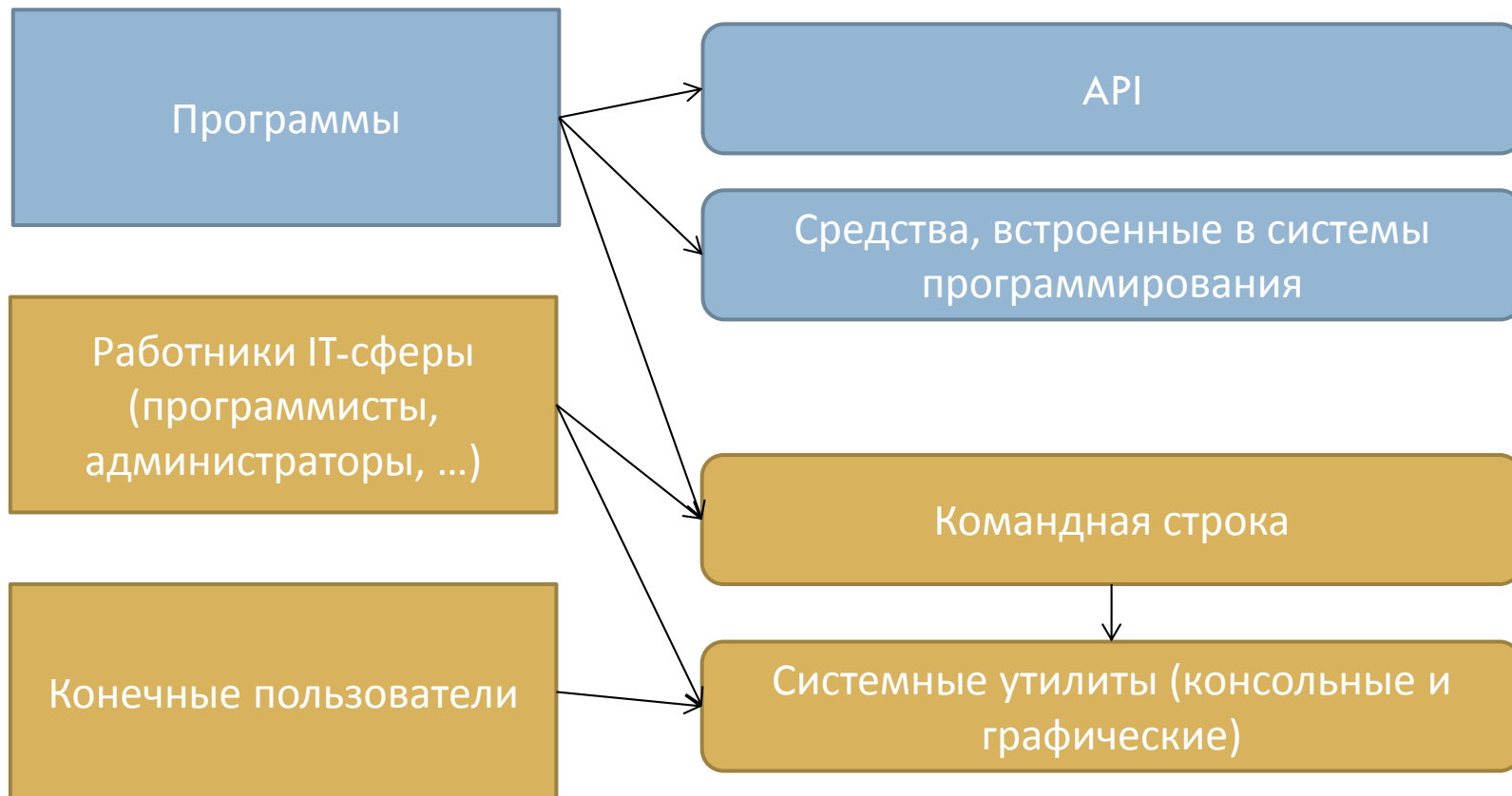
РАБОТА С КОМАНДНОЙ СТРОКОЙ

Что такое командная строка?



- Командная строка является одним из элементов интерфейса между операционной системой и людьми – пользователями этой операционной системы. Этот элемент интерфейса присутствует практически во всех ОС.

Уровни интерфейса между ОС и пользователями



Когда удобно использовать командную строку?

- невозможность работать в графической оболочке по каким-либо причинам;
- переименование большого количества однотипно названных файлов;
- смена настроек многих пользователей;
- другие рутинные работы по настройке системы.

Как выполняется работа в командной строке?

Работа в командной строке выполняется в консоли. Система выводит на экран *приглашение*, в ответ на которое пользователь вводит с клавиатуры *команду* операционной системы. Команда выполняется (при этом возможен диалог с пользователем), после чего на экране вновь появляется *приглашение*.

Как запускается командная строка?

1. Пуск → Все программы → Стандартные → Командная строка
2. Win+R → набрать CMD в открывшемся окне
3. Использовать файловые менеджеры (Far, Total Commander)

Менеджеры командной строки (командные процессоры)

1. Для Windows:

- COMMAND.COM (в настоящее время используется только для старых 16-битовых приложений)
- CMD.EXE
- Power Shell

2. Для Linux:

- bash

Классификация команд

Команды Windows делятся на *внутренние* и *внешние*.

Внутренние команды обрабатываются самим командным процессором, внешние – это вызов каких-то приложений. Разница в формате отсутствует, но, если, например, команда `сору` будет обработана всегда, то команда `format` – в зависимости от доступности соответствующего приложения. Если доступен и исполняемый файл с именем внутренней команды, то при правильных значениях параметров будет работать внутренняя команда, при неверных – исполняемый файл.

Формат команд

Команды операционной системы имеют следующий формат:

имякоманды [параметры]...

(здесь и далее синий цвет означает запись метасимволов в нормальной форме Бэкуса-Наура). Имя команды и её параметры отделяются пробелами. Если пробел является частью имени команды или параметра, такая конструкция обрамляется двойными кавычками.

В составе имени команды и параметров недопустимы *специальные* символы:

&<> () [] { } ^ = ; ! ' + , ` ~

<Отступление 1> Файловая система

Термином «файловая система» зачастую обозначают три различных понятия:

- Логический способ организации данных на внешних устройствах, обеспечивающий простой и понятный доступ к каждому элементу данных (файлу);
- Физический способ организации данных на внешних устройствах;
- Часть API, обеспечивающая работу с данными на внешних устройствах

Мы будем использовать первое понятие

Файловая система Windows



Файловая система Windows имеет строго иерархическую структуру.

В состав файловой системы Windows входят следующие уровни иерархии:

- диски и сетевые подключения;
- папки (директории, каталоги);
- файлы

Диски

Диск (с точки зрения файловой системы) представляет собой одно из деревьев иерархического леса. Диски обозначаются буквами латинского алфавита, за которыми ставится двоеточие, например, D :

В качестве дисков могут выступать:

- жёсткие диски компьютера или результаты их разбиения на несколько частей;
- память подключаемых внешних устройств;
- виртуальные диски, формируемые некоторыми программами (Daemon tool, Alcohol 120 и т.д.);
- виртуальные диски, подключаемые командами Windows (SUBST, NET USE и т.д.)

На каждом диске создаётся корневая папка, обозначаемая \

Сетевые подключения

Сетевые подключения обозначаются как

`\\имя ресурса`

(ресурсы также могут иметь иерархическую структуру)

Пример сетевого подключения:

`\\serv-stud\subfaculty`

Для каждого подключения также создаётся корневой каталог \

Старые приложения не поддерживают прямого доступа к сетевым подключениям, для них необходимо создавать виртуальный диск командой `net use`!

Подпапки и файлы

В каждой папке, начиная с корневой, могут быть созданы подпапки и файлы. В каждой подпапке также могут быть подпапки и файлы следующего уровня.

Имена файлов и подпапок должны быть уникальными в рамках одной папки.

Полный путь к файлу или папке начинается с корневой папки диска или сетевого подключения и выглядит, например, так:

`D:\Lectures\2015-2016 учебный год\2 семестр\1`

В этом примере нельзя однозначно сказать, чем является «1» – файлом или папкой!

Собственное имя и расширение

Если в имени файла или папки присутствует хотя бы одна точка, часть имени до последней точки считается *собственным именем*, а часть после неё – *расширением* файла (папки).

Как собственное имя, так и расширение может отсутствовать.

Примеры:

tempfile – файл без расширения

.idea – файл без имени, но с расширением

Расширение файла позволяет определить, какая программа используется по умолчанию для его обработки.

Шаблоны имён файлов

Если в имени файла присутствуют специальные символы * и/или ?, такое имя считается *шаблоном*.

Шаблоны используются для решения следующих задач:

- Проверить, удовлетворяет ли имя файла шаблону;
- Найти (обработать) все файлы, чьи имена удовлетворяют шаблону

В шаблонах * – заменяет произвольное количество любых символов, ? – заменяет один произвольный символ.

Шаблоны имён файлов (продолжение)

Примеры шаблонов:

* , *.*	Все файлы
*.	Все файлы без расширения
??t.txt	Например, 01t.txt, но не 2t.txt
??*.??	Файлы, содержащие не менее двух символов в собственном имени и ровно два символа в расширении
g.	Файлы, собственное имя которых заканчивается на g

Текущий диск и текущая папка

Один из дисков файловой системы является *текущим*. Одна из папок каждого диска является *текущей*.

Вместо полного пути к файлу можно указывать путь относительно текущего диска и/или папки, например:

<code>e:myfile.in</code>	Файл находится в текущей папке диска e:
<code>temp\myfile.in</code>	Файл находится в подпапке temp текущей папки текущего диска
<code>.</code>	Обозначение текущей папки текущего диска
<code>..\myfile.in</code>	Файл находится в папке, чьей подпапкой является текущая папка текущего диска
<code>\myfile.in</code>	Файл находится в корневой папке текущего диска

Характеристики и атрибуты файлов

Файлы обладают рядом **характеристик**:

- ❑ дата создания файла;
- ❑ время создания и редактирования файла;
- ❑ длина (объем) файла.
- ❑ атрибуты файла, которые указывают на характер его использования и возможность доступа к нему:
 - **Read-Only**— файл, использующийся только для чтения; чаще всего он не может быть уничтожен или отредактирован, но допустимо создание копии и операций над ней;
 - **Archive**— архивный, этот атрибут устанавливается при успешном закрытии файла;
 - **Hidden**— скрытый файл;
 - **System**— системный.

</Отступление 1>



Ключи команд

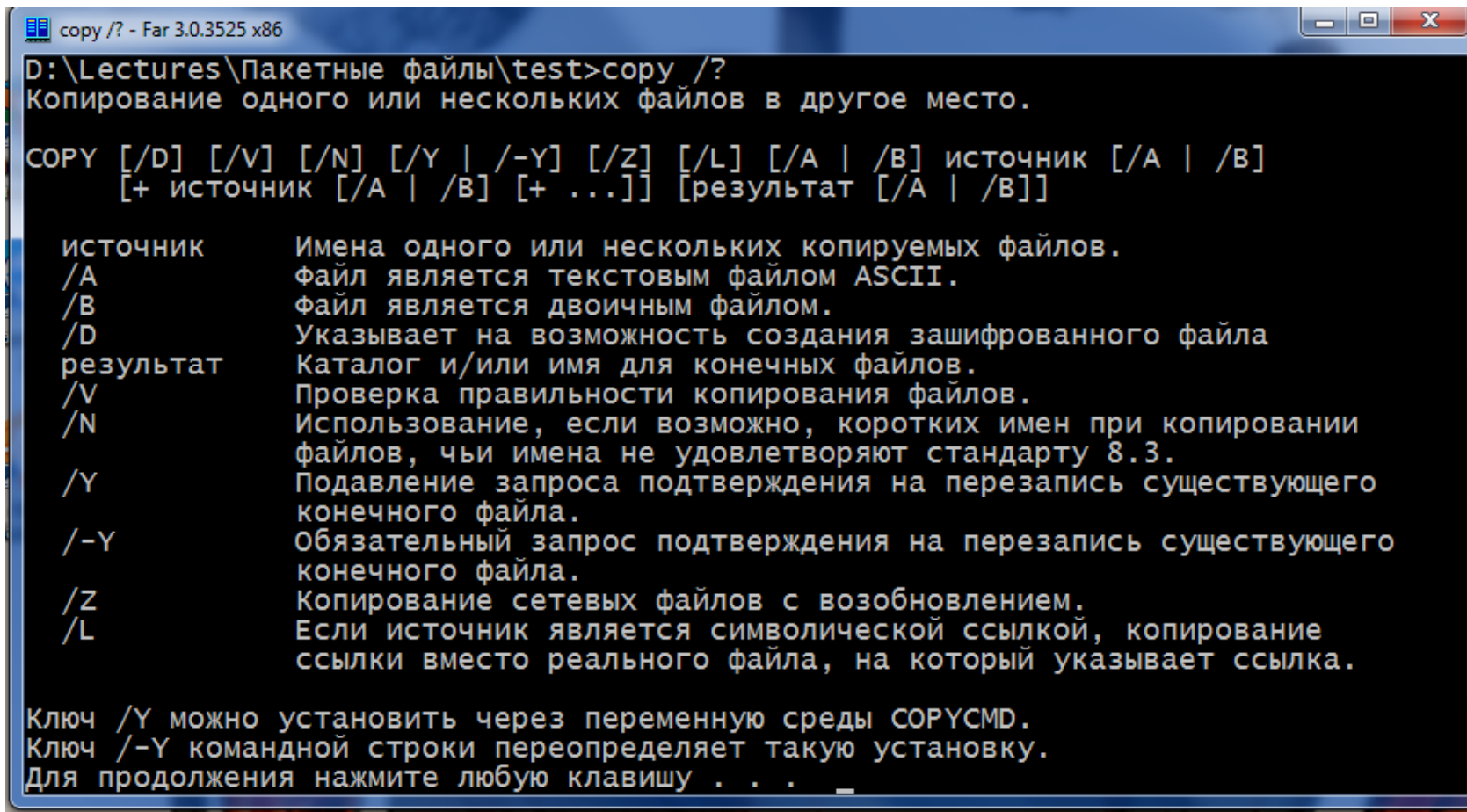
Некоторые команды в качестве одного или нескольких параметров имеют *ключи*.

Использование ключей позволяет изменить режим работы команды.

Ключи обычно начинаются с символа */*, за которым следует название ключа. В частности, ключ */?* выводит в окно командной строки справку о работе с командой.

Значения некоторых других ключей будут пояснены при рассмотрении соответствующих команд.

Пример работы ключа /?



```
copy /? - Far 3.0.3525 x86
D:\Lectures\Пакетные файлы\test>copy /?
Копирование одного или нескольких файлов в другое место.

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

источник      Имена одного или нескольких копируемых файлов.
/A            файл является текстовым файлом ASCII.
/B            файл является двоичным файлом.
/D            Указывает на возможность создания зашифрованного файла
результат     Каталог и/или имя для конечных файлов.
/V            Проверка правильности копирования файлов.
/N            Использование, если возможно, коротких имен при копировании
              файлов, чьи имена не удовлетворяют стандарту 8.3.
/Y            Подавление запроса подтверждения на перезапись существующего
              конечного файла.
/-Y           Обязательный запрос подтверждения на перезапись существующего
              конечного файла.
/Z            Копирование сетевых файлов с возобновлением.
/L            Если источник является символической ссылкой, копирование
              ссылки вместо реального файла, на который указывает ссылка.

Ключ /Y можно установить через переменную среды COPYCMD.
Ключ /-Y командной строки переопределяет такую установку.
Для продолжения нажмите любую клавишу . . . _
```

Потоки ввода/вывода

Потоки используются в командах для унифицированного распределения и перенаправления ввода/вывода. В следующей таблице представлены стандартные потоки, по умолчанию связанные с консолью :

Поток	Номер дескриптора	Описание
STDIN	0	Стандартный ввод (только чтение)
STDOUT	1	Стандартный вывод (только запись)
STDERR	2	Вывод ошибок (только запись)

Стандартные устройства

Вместо имён файлов в командах Windows могут использоваться следующие стандартные устройства:

Устройство	Описание
con	Консоль (может быть использована как для ввода, так и для вывода)
nul	Пустое устройство (при вводе сразу же наступает ситуация «конец файла», при выводе данные растворяются в воздухе)
prn	Принтер (в настоящее время практически не используется)

Простейшая команда - echo

Команда `echo` позволяет вывести на консоль содержимое своих операторов или переопределить режим вывода на консоль текстов последующих команд в командных файлах:

Команда	Результат
<code>echo hello world!</code>	На консоль будет выведена строка «hello world!»
<code>echo on</code>	Включает вывод на консоль текстов последующих команд
<code>echo off</code>	Выключает вывод на консоль текстов последующих команд
<code>@echo off</code>	Выключает вывод на консоль текст а этой и последующих команд
<code>echo (или echo.)</code>	Выводит текущее значение режима вывода на экран

Другие команды для работы с консолью

Команда	Описание
pause	Приостанавливает работу программы или пакетного файла до нажатия любой клавиши
cls	Очищает окно консоли
color	Изменяет цвет шрифта и фона при последующем выводе на консоль
more файл	Выводит на консоль файл, выполняя задержку после заполнения одного экрана
команда more	Выполняет задержку после заполнения одного экрана при выводе результатов указанной команды

Команды для работы с файловой системой

Команда	Описание
Диск (например, d:)	Смена текущего диска
cd папка [/d]	Смена текущей папки (и диска при указании ключа /d)
md [диск] папка	Создаёт новую папку, включая, при необходимости, промежуточные папки
copy источник [+ источник1] ... приемник [/y]	Копирование файлов (возможно, с их сцеплением). Ключ /-y подавляет запрос на перезапись
ren файл новоеимя	Переименование одного или нескольких файлов. Если файл с новым именем существует, команда отвергается
move файл новоеместо	Перенос одного или нескольких файлов в новое место. Если переносится один файл, можно одновременно и переименовать его

Команды для работы с файловой системой (продолжение)

Команда	Описание
del файлы erase файлы	Удаление одного или нескольких файлов (папок)
subst диск папка subst диск /D	Создаёт виртуальный диск на основе указанной папки. Ключ /D уничтожает ранее созданный виртуальный диск
tree [папка]	Выводит на консоль дерево папок, начиная с указанной (или с текущей, при отсутствии параметра)
dir [папка]	Выводит содержимое указанной папки
fc файл1 файл2 [/B]	Сравнивает содержимое двух файлов (бинарных, при указании ключа /B)
find строка файлы	Поиск текстовой строки в файлах

Вопрос...

Как сработают следующие команды copy?

- `copy c:\boot.ini c:\boot.bak`
- `copy c:\boot.ini d:`
- `copy a.txt+b.txt c.txt`
- `copy a:*.*`
- `copy con a.txt`
- `copy nul nul`
- `copy *.in *.out /-y`

Несколько команд в одной строке

Иногда имеет смысл сгруппировать в одной строке несколько команд, которые должны быть выполнены последовательно и независимо друг от друга. В этом случае мы можем отделить одну команду от другой символами или парами символов `&`, `&&`, `| |`, а также брать их в скобки. Пример такого объединения:

```
md \a & cd \a & md b & cd b & md c & cd c & md d
```

Несколько команд в одной строке (продолжение)

Различия в применении символов-объединителей

<code>команда1 & команда2</code>	Выполняется сначала команда1, а затем – команда2
<code>команда1 && команда2</code>	Команда2 выполняется только после успешного (т.е. с нулевым кодом) завершения команды1.
<code>команда1 команда2</code>	Команда2 выполняется только после неудачного (т.е. с ненулевым кодом) завершения команды1.
<code>(команда1 & команда2) (команда3)</code>	Команда1 и команда2 выполняются последовательно, а команда3 – только после неудачного завершения команды2.

<Отступление 2> Коды завершения программ

Любые приложения могут информировать операционную систему о результатах своей работы, формируя *код завершения* (код возврата) – целое неотрицательное число.

```
int main () {  
...  
    return 1; // код завершения - 1  
...  
    return 0; // код завершения - 0  
}
```

Код завершения программ, описанных как `void main()`, зависит от компилятора (в Visual Studio 2008, 2010, 2013 это 0)

Коды завершения программ (продолжение)

Чаще всего код завершения формируется по следующему принципу: при нормальном завершении программы она возвращает 0, а ненулевые коды завершения говорят о возникших проблемах. Причём чем серьёзнее проблемы, тем больше код завершения.

Поэтому иногда код завершения называется уровнем ошибки (error level).

Этот принцип необязателен! Так, программа `fc` возвращает 0, если сравниваемые файлы совпадают, и 1 в противном случае.

</Отступление 2>



Перенаправление ввода-вывода

Команды, как и приложения, могут работать со стандартными дескрипторами файлов, описанными ранее. Изначально эти дескрипторы связаны с экраном или окном консоли. Однако их можно переопределить.

Для переопределения дескриптора `stdin` надо в конце команды написать конструкцию

< [ИСТОЧНИК](#)

Пример переопределения ввода:

```
my_prog <input.txt
```

и теперь приложение `my_prog.exe` будет читать данные из файла вместо консоли, даже если в ней нет вызова функции `freopen`.

Перенаправление ввода-вывода (продолжение)

Для переопределения дескриптора `stdout` надо в конце команды написать одну из конструкций

> приёмник

>> приёмник

Первая конструкция создаёт новый приёмник, вторая – дописывает в конец существующего.

Примеры переопределения вывода:

```
my_prog >output.txt
```

и теперь приложение `my_prog.exe` будет выводить данные в файл вместо консоли.

```
copy 01.in input.txt >nul
```

Сообщения о результатах работы команды `copy` (количестве скопированных файлов) будут подавлены.

Перенаправление ввода-вывода (продолжение примеров)

```
dir c: >dirs.txt  
dir d: >>dirs.txt  
dir e: >>dirs.txt
```

Эти три команды, выполненные одна за другой, позволяют получить в одном файле информацию о содержимом текущих каталогов трех дисков.

Перенаправление ввода-вывода (продолжение)

Для переопределения дескриптора `stderr` надо в конце команды написать одну из конструкций

```
2> приёмник
```

```
2>> приёмник
```

```
2>&1
```

Последняя из приведенных записей позволяет перенаправить сообщения об ошибках в то же место, что и стандартный вывод.

Пример переопределения вывода для `stderr`:

```
my_prog >out.txt 2>&1
```

```
my_prog >out.txt 2>out.txt
```

Нам необходимо направить в один и тот же файл как стандартный вывод, так и сообщения об ошибках. В первом случае показан правильный, а во втором – неправильный путь решения этой задачи.

Конвейеры

Можно записать несколько команд в одной строке, так что стандартный вывод предыдущей команды будет перенаправлен в качестве стандартного ввода последующей команды. На консоль при этом ничего выводиться не будет.

Для реализации конвейера необходимо записать конструкцию

`команда1 | команда2`

Примеры использования конвейеров:

```
echo y | del *.* >nul
```

При удалении всех файлов команда `del` выдает запрос на подтверждение такой операции. Для подавления этого запроса, если мы уверены в правильности действий, можно принудительно направить результат работы команды `echo` (суть которой, как известно, состоит в выводе на консоль своих параметров) в качестве ответа на запрос.

В настоящее время пример устарел, поскольку в команде `del` появился ключ `/q`, подавляющий запросы на подтверждение.

Конвейеры (продолжение)

Некоторые команды (например, рассмотренные ранее команды `more` и `find`) специально предназначены для работы с конвейерами.

```
g++ --help | more
```

Результаты вывода команды `g++` (информация о возможных режимах компиляции GNU C++) будут выводиться с задержкой.

```
dir | find ".asm" > dirasm.txt
```

В отличие от команды `dir *.asm > dirasm.txt`, в файл не выводится служебная информация, генерируемая командой `dir`.

Командные файлы

Командные файлы (сценарии командной строки, пакетные файлы, в просторечии - батники) представляют собой текстовые файлы, в состав которых входят команды операционной системы и специальные команды, служащие для организации управления программным потоком. Командные файлы имеют расширение `bat` или (начиная с Windows 2000) `cmd`. Командные файлы создаются в любом текстовом редакторе. После того, как командный файл создан, он может быть запущен, как и другое выполняемое приложение.

Выполнение командных файлов

Выполнение командного файла происходит в режиме *интерпретации*: каждая строка считывается и тотчас выполняется. Это, в частности, приводит к тому, что не выдается сообщений даже о командах с ошибочным синтаксисом, если эти команды не будут считаны при текущем запуске командного файла.

Пример:

```
ren *.in *.i  
exit  
copi res.txt ..
```

Вторая строка выполняет выход из пакетного файла, поэтому сообщения о неправильной третьей команде не будет.

Пример простейшего командного файла

Для запуска на выполнение программы, написанной на Ассемблере, необходимо выполнить ее компиляцию, затем линковку, и наконец, запустить на выполнение получившийся файл. Можно создать командный файл с именем `my_prog.bat`, объединяющий все эти этапы для программы `my_prog.asm`:

```
tasm my_prog  
tlink my_prog  
my_prog
```

Этот пример выглядит убого – он предназначен для работы с конкретным файлом, не выполняется проверка результатов предыдущих шагов. В дальнейшем этот пример будет переработан.

Комментарии в командных файлах

Два способа задать комментарии:

- записать специальную команду `rem`

`rem` пример батника на компиляцию, линковку, выполнение

- поставить двоеточие в начале строки, формируя *метку*

```
:tlink my_prog
```

Передача параметров в командные файлы

При работе с командными файлами можно организовать передачу в них параметров командной строки так же, как и в любые другие приложения.

Первые девять формальных параметров обозначаются в пакетном файле конструкциями %1 – %9. Конструкция %0 позволяет получить полное имя запускаемого командного файла.

Передача параметров в командные файлы

При работе с командными файлами можно организовать передачу в них параметров командной строки так же, как и в любые другие приложения.

Первые девять формальных параметров обозначаются в пакетном файле конструкциями %1 – %9. Конструкция %0 позволяет получить полное имя запускаемого командного файла.

Пример передачи параметров в командные файлы

Изменим предыдущий пример так, чтобы командный файл (дадим ему имя `asm.bat`) позволял работать с любым именем исходного файла. Для этого имя исходного файла будем передавать в качестве единственного параметра:

```
tasm %1  
tlink %1  
%1
```

Для работы с файлом `my_prog.asm` достаточно запустить строку

```
asm my_prog
```

Изменение представления параметров

Можно изменить представление параметров в пакетном файле с помощью следующих синтаксических конструкций:

%~1	из переменной %1 удаляются обрамляющие кавычки (")
%~f1	переменная %1 расширяется до полного имени файла
%~d1	из переменной %1 выделяется только имя диска
%~p1	из переменной %1 выделяется только путь к файлу
%~n1	из переменной %1 выделяется только имя файла
%~x1	из переменной %1 выделяется только расширение имени файла (вместе с точкой)
%~t1	переменная %1 расширяется до даты /времени последней модификации файла
%~z1	переменная %1 расширяется до размера файла
%~nx1	из переменной %1 выделяется только имя и расширение файла

Пример изменений в параметрах командных файлов

Продолжим рассмотрение командного файла `asm.bat`. При работе с ним возможны ошибки, если в качестве параметра передать не имя программы, а имя исходного файла (например, `my_prog.asm`). Чтобы дать возможность использовать любой формат параметра, следует изменить наш командный файл:

```
tasm %~n1  
tlink %~n1  
%~n1
```

Можно также явно указать имена обрабатываемых файлов:

```
tasm %~n1.asm  
tlink %~n1.obj  
%~n1.exe
```

Как получить произвольное количество параметров?

Команда `shift` позволяет удалить значение параметра `%1` и сдвинуть последующие параметры (значение `%1` заменяется на `%2`, ..., появляется новое значение параметра `%9`).

Управление программным потоком

Команды в командных файлах, рассмотренных ранее, выполнялись последовательно, друг за другом. Если нам необходимо изменить порядок выполнения команд, надо использовать специальные команды управления программным потоком.

Мы будем рассматривать следующие команды управления:

- **goto**
- **call**
- **exit**
- **goto :eof**
- **if**
- **for**

Команда `goto`

Команда

`goto метка`

позволяет перейти к строке, перед которой стоит специальная команда-метка

`:метка`

Обратим внимание на то, что вследствие режима интерпретации не проводится контроль на наличие меток, к которым нет обращений, и среди таких меток допускаются дубликаты. Поэтому один из удобных способов закомментировать команду в командном файле – поставить перед ней двоеточие и превратить тем самым команду в неиспользуемую метку.

Необходимость в команде `goto`

Оператор `goto` считается неудобным, поскольку он резко снижает читабельность программы. В некоторых языках (например, в JAVA) он запрещён. Однако иногда без него не обойтись...

В настоящее время считается допустимым использование `goto` в следующих ситуациях:

- выход из нескольких вложенных циклов;
- отсутствие в синтаксисе языка конструкции `else` (в ранних версиях командных файлов)

Вызов командных файлов и выход из них

Командные файлы могут быть вызваны:

- из командной строки (или из приложения);
- из другого командного файла.

Если командный файл вызван из командной строки, он может быть завершён следующими способами:

- после выполнения последней команды пакетного файла;
- командой `exit`;
- командой `goto :eof` (Здесь `eof` – фиктивная метка, которая может отсутствовать. Она якобы находится в строке, следующей за последней строкой пакетного файла).

Вызов командных файлов и выход из них (продолжение)

Если командный файл вызывается из другого командного файла, различают *мягкий* и *жёсткий* вызов и выход.

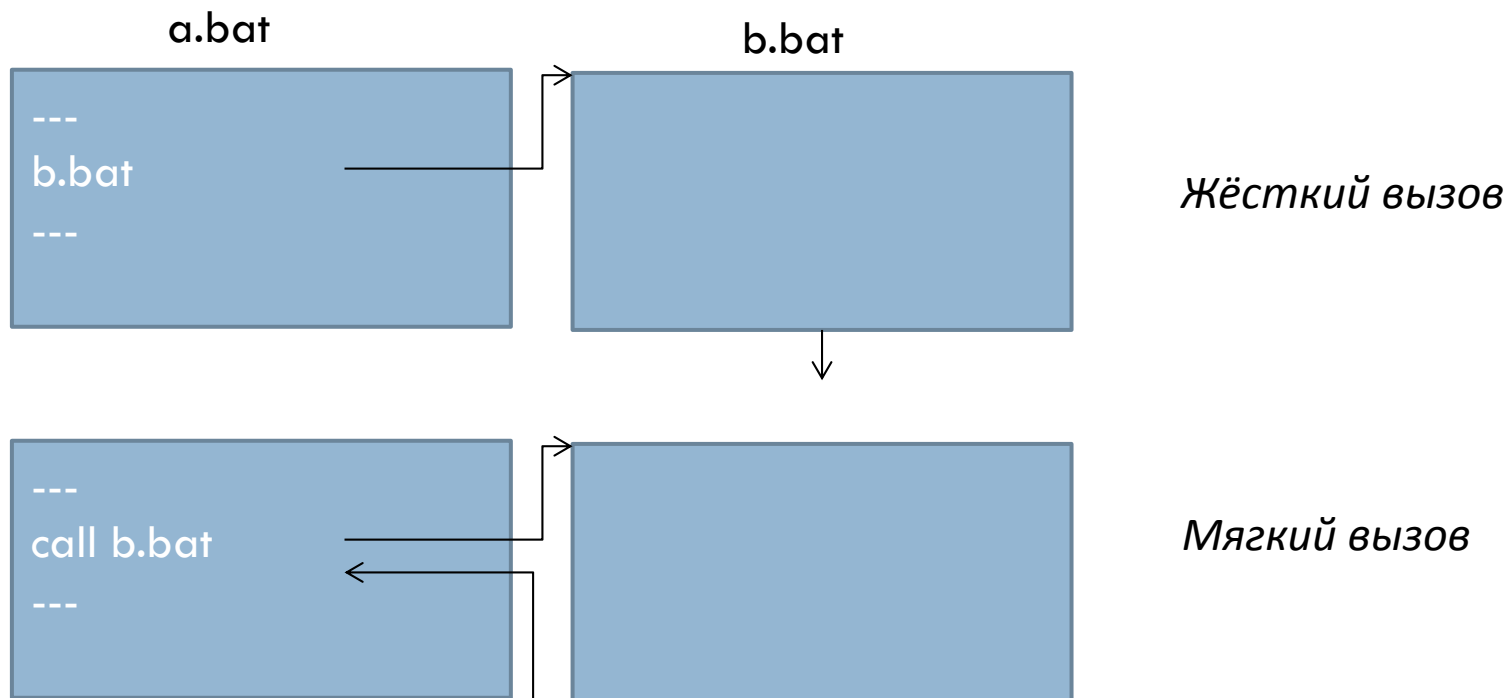
При мягком вызове и выходе после окончания работы вызванного пакетного файла управление передаётся в вызывающий файл.

При жёстком вызове или жёстком выходе управление передаётся в командную строку. Оставшиеся команды вызывающего файла игнорируются.

Жёсткий и мягкий вызов командных файлов

Для жёсткого вызова пакетного файла достаточно указать его имя (и при необходимости, параметры) в командной строке.

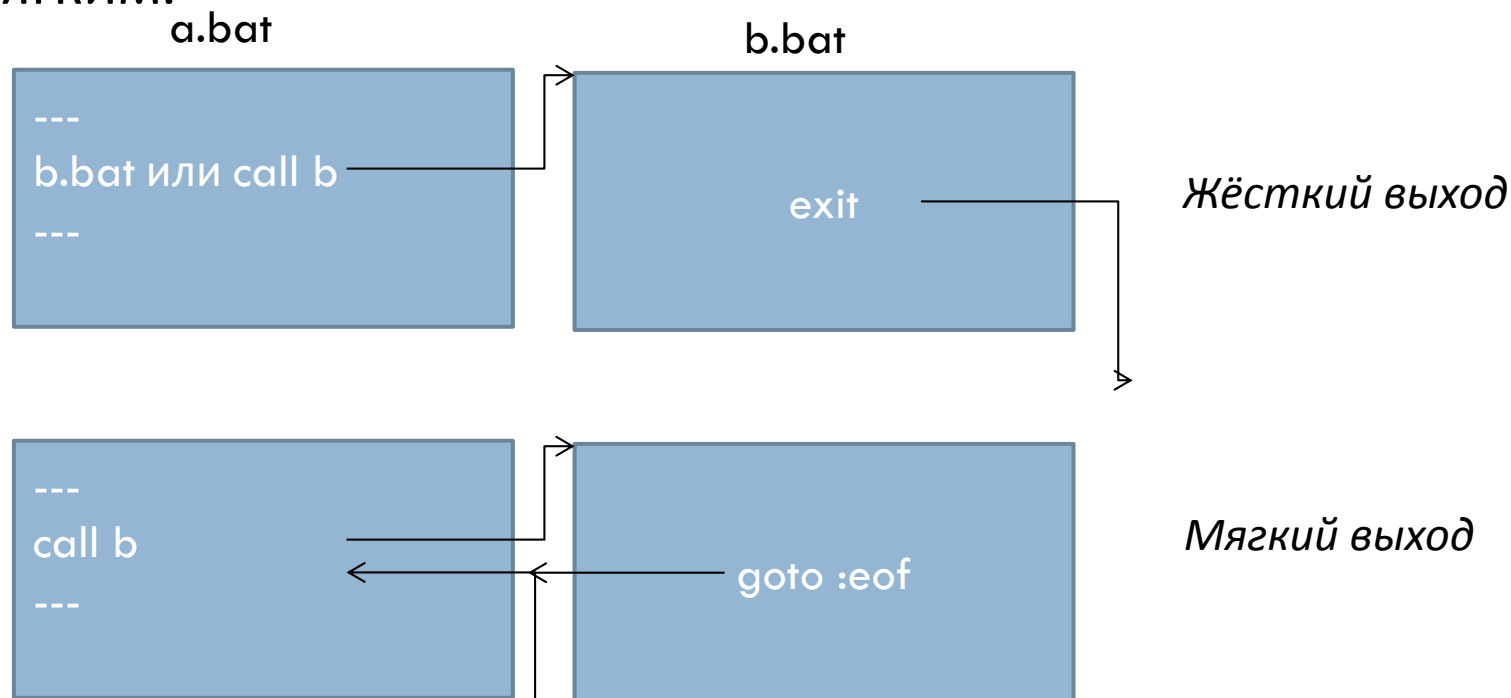
Для мягкого вызова необходимо выполнить команду call.



Жёсткий и мягкий выход из командных файлов

Для жёсткого выхода из пакетного файла необходимо записать команду `exit`.

Другие способы завершения командного файла делают выход **МЯГКИМ**.



Написание псевдоподпрограмм

Параметром команды `call` может быть не только имя внешнего командного файла, но и метка в том же пакетном файле. Таким образом, мы можем организовать пакетный файл из нескольких частей (псевдоподпрограмм).

Такой файл имеет следующую структуру:

```
@echo off
...
call :p1
...
goto :eof
:p1
...
call :p2
...
goto :eof
```

```
:p2
...
:eof
```

Команда if – обычный режим

В обычном режиме команда if имеет три формата, в каждом из которых имеется вложенная команда. Обратите внимание на отсутствие конструкции else в этой команде!

if [not] строка1 == строка2 команда

if [not] exist файл команда

if errorlevel число команда

Команда if – обычный режим (продолжение)

if [not] строка1 == строка2 команда

Этот формат позволяет выполнить вложенную команду только в том случае, когда две строки совпадают (или не совпадают в случае фразы not). Обычно в одной или обеих строках записываются параметры командного файла или их комбинации.

Так, проверка на наличие первого параметра может выглядеть как

```
if %1.==. команда
```

Вместо точки может стоять любой символ.

Команда if – обычный режим (продолжение)

if [not] строка1 == строка2 команда

Рассмотрим пример этой конструкции. Пусть нам надо выполнить команду `dir` для всех параметров командного файла (их количество неограничено). Соответствующий командный файл будет иметь следующий вид:

```
@echo off
:next
if %1.==. goto :eof
dir %1
shift
goto :next
```

Команда if – обычный режим (продолжение)

if [not] exist файл команда

Этот формат команды if позволяет выполнить вложенную команду только в том случае, когда существует (или не существует в случае фразы not) указанный файл или группа файлов, объединенная шаблоном. Приведем пример этой конструкции:

```
if exist *.bak del *.bak >nul
```

Команда if – обычный режим (продолжение)

if errorlevel число команда

Третий формат команды if позволяет выполнить вложенную команду в зависимости от кода завершения, который возвратила предыдущая команда.

Вложенная команда выполняется, если код завершения больше либо равен указанному числу! Поэтому проверку нескольких кодов завершения целесообразно выполнять начиная с бóльших значений:

```
if errorlevel 2 goto m2  
if errorlevel 1 goto m1
```

Использование if в примере asm.bat

```
@echo off
if %1.==. goto no_parm
if not exist %~n1.asm goto no_file
tasm %~n1
if errorlevel 1 goto :eof
tlink %~n1
if errorlevel 1 goto :eof
%~n1
goto :eof

:no_parm
echo Формат вызова: asm имя_исходного_файла
goto :eof

:no_file
echo Файл %~n1.asm не найден!
:eof
```


Команда for - обычный режим

Формат команды for в обычном режиме:

for %переменная **in** (набор) **do** команда

Здесь:

- переменная – буква латинского алфавита (прописные и строчные буквы различаются)
- (набор) - набор, состоящий из одного или нескольких файлов. допускается использование подстановочных знаков)
- команда - команда, которую следует выполнить для каждого файла. Переменная может использоваться в параметрах этой команды.

Примеры использования команды for

```
for %i in (proc1, proc2, proc3) do call asm.bat %i
```

(запускается последовательная компиляция трёх программ)

Запятая в этом примере может отсутствовать!

```
for %k in (*.i) do ren %~nk.i 0%~nk.in
```

(сложное переименование нескольких файлов)

```
for %i in (bak old tmp) do del *.%i
```

(быстрое удаление файлов с разными расширениями)

Особенности записи команды for в командных файлах

Если команда for записывается в командном файле, при записи ее параметра во вложенной команде символ '%' должен дублироваться. Так, предыдущая команда, если она входит в состав командного файла, должна быть записана так:

```
for %%i in (bak old tmp) do del *.%%i
```

Вопрос

Что делает следующий командный файл?

```
copy nul alltext
```

```
for %%i in (*.txt) do copy alltext+%%i alltext
```

Расширенный режим – блоки

Блоком называется совокупность из нескольких команд, взятая в круглые скобки.

Существуют два формата блоков:

- (команда)
- (
 команды ...
)

Блоки используются в расширенном режиме для команд `if` и `for`.

Пример использования блока в команде for

@echo off

```
for /d %%i in (E:\profiles\*.*) do (  
  for %%j in (1, 2, 3) do (  
    if exist %%i\?????t%%j.cpp copy %%i\?????t%%j.cpp sources_all >nul  
    if exist %%i\?????t%%j.gpp copy %%i\?????t%%j.gpp sources_all >nul  
    if exist %%i\?????t%%j.pas copy %%i\?????t%%j.pas sources_all >nul  
    if exist %%i\?????t%%j.dpr copy %%i\?????t%%j.dpr sources_all >nul  
  )  
  for /d %%j in (%%i\?????t4) do (  
    md 4\contestants\%%~nj  
    copy %%i\%%~nj\*. * 4\contestants\%%~nj >nul  
  )  
)
```

Расширенный режим – дополнения в команде for

При использовании расширенного режима обработки команд команда for получает дополнительные ключи, которые существенно расширяют ее возможности. Ключи команды for записываются непосредственно после её названия и нечувствительны к регистру.

FOR /D %переменная IN (набор) DO команда

Если набор содержит подстановочные знаки, команда выполняется для всех подходящих имен каталогов, а не имен файлов.

Пример использования этого ключа показан на предыдущем слайде.

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

Выполнение команды для каталога [диск:]путь, а также для всех подкаталогов этого пути. Если после ключа /R не указано имя каталога, используется текущий каталог. Если набор - это одиночный символ точки (.), команда просто перечисляет дерево каталогов.

Расширенный режим – дополнения в команде `for`, ключ `/L`

Команда

`for /L %переменная in (начало, шаг, конец) do команда`
работает подобно оператору `for` во многих языках программирования – значение переменной является числовым и пробегает весь заданный промежуток.

Пример:

Пусть создана галерея из 125 графических файлов `1.jpg, ..., 125.jpg`. Нумерация файлов соответствует последовательности, в которой их необходимо просматривать. Однако обычная лексикографическая упорядоченность не дает выполнить просмотр в режиме слайд-шоу. Переименуем файлы с тем, чтобы добавить к имени ведущие нули:

```
for /L %i in (1,1,9) do ren %~ni.jpg 00%~ni.jpg
for /L %i in (10,1,99) do ren %~ni.jpg 0%~ni.jpg
```


Расширенный режим – дополнения в команде `for`, ключ `/F`

Команда

`for /F ["правила"] %переменная in (файлы) do команда`

выполняет построчное чтение файлов, указанных во фразе `in`, разбор каждой строки на подстроки (токены) в соответствии с записанными правилами и передачу полученных токенов во вложенную команду. Токены нумеруются, начиная с единицы. Для каждого токена, начиная со второго, создается дополнительная односимвольная переменная, имя которой строится исходя из имени параметра команды `for`: если параметр `for` имеет имя `%i`, то значение первого токена помещается в переменную `%i`, значение второго – в переменную `%j` и т.д.

Расширенный режим – дополнения в команде for, ключ /F (продолжение)

Правила для обработки токенов представляют собой заключенную в кавычки строку, содержащую следующие ключевые слова, разделенные пробелами:

eof=символ	строки, начинающиеся с указанного символа считаются комментариями, и не будут обрабатываться
skip=число	число строк в начале файла, которые не будут обрабатываться
delims=символы	определение дополнительных символов, являющихся разделителями между токенами. По умолчанию разделителями являются пробел и знак табуляции.
tokens=список чисел	определение номеров токенов, выделяемых из каждой строки файла и передаваемых для выполнения в тело цикла. Номера токенов разделяются запятой или дефисом. Если последний символ в строке tokens= является звездочкой, создается дополнительная переменная, значением которой будет весь оставшийся текст в строке.

Расширенный режим – дополнения в команде for, ключ /F (пример)

Пусть нам необходимо создать несколько новых пользователей, назначив каждому из них сгенерированный случайным образом пароль и запретив пользователю изменять этот пароль. Для выполнения этих действий используется команда NET USER в следующем формате:

```
net user имяпользователя пароль /add /passwordchg no
```

Программа генерации паролей поместила результаты своей работы в текстовый файл pswd.txt. Каждая строка этого файла содержит имя пользователя и пароль, разделенные пробелом. Тогда выполнить указанные действия можно командой

```
for /f "tokens=1,2" %a in (pswd.txt) do (  
    net user %a %b /add /passwordchg no  
)
```

Расширенный режим – дополнения в команде if

Первое и очень важное дополнение – появление конструкции `else`.

Теперь команда `if`, помимо рассмотренных ранее, имеет следующие форматы:

```
if условие (команда1) else команда2
```

или

```
if условие (  
    команды...  
) else (  
    команды...  
)
```

Пример:

```
if exist *.bak (del *.bak) else echo Файл не существует!
```

Расширенный режим – дополнения в команде if (сравнение строк)

`if [/I] строка1 операторсравнения строка2 команда`

В качестве оператора сравнения могут быть следующие конструкции:

- `EQ` – равно;
- `NEQ` - не равно;
- `LSS` – меньше;
- `LEQ` - меньше или равно;
- `GTR` – больше;
- `GEQ` - больше или равно.

Ключ `/I` задает сравнение текстовых строк без учета регистра. Его можно также использовать и в форме `строка1==строка2` команды `if`.

Сравнения проводятся по общему типу данных, так что если обе строки содержат только цифры, то они преобразуются в числа, после чего выполняется сравнение чисел.

Расширенный режим – дополнения в команде if (использование %errorlevel%)

В расширенном режиме можно использовать конструкцию %ERRORLEVEL%, которая будет развернута в строковое представление текущего значения кода завершения. Например, с помощью данной строки можно выполнить следующее:

```
goto answer%ERRORLEVEL%
:answer0
echo Получен код возврата 0
goto m
:answer1
echo Получен код возврата 1
:m
```

Допускается и применение операторов числового сравнения:

```
if %ERRORLEVEL% EQL 1 goto warnings
```

<Отступление 3> Переменные окружения

Понятие «переменная окружения» («переменная среды», «environment variable») появилось во времена MS-DOS, когда требовалось передать какие-то параметры запускающейся программе или организовать связь между двумя последовательно выполняющимися программами.

Для реализации этих задач была придумана концепция переменных окружения – глобальных объектов, характеризующихся именем и строковым значением. Каждое запущенное приложение (равно как и пользователь-человек) имеет полный доступ к переменным, т.е. оно может читать, создавать, изменять значение, удалять любую переменную.

Для работы с переменными окружения используется команда

```
set имяпеременной=значениепеременной
```

Для создания новой или изменения значения существующей переменной надо указать ее значение. Отсутствие значения приведет к уничтожению переменной.

Переменные окружения (продолжение)

Начиная с Windows 2000, в концепцию переменных окружения внесены существенные изменения. прежде всего, разграничен доступ к переменным за счет разделения их на три категории:

- системные переменные;
- пользовательские переменные;
- локальные переменные

Системные переменные окружения

Системные переменные создаются и модифицируются только пользователями, имеющими полномочия администратора. Другие пользователи могут лишь читать значения этих переменных. При этом единожды установленные значения системных переменных остаются неизменными для всех пользователей.

Рассмотрим некоторые из системных переменных окружения.

- ❑ Переменная `path` задает список путей, в которых необходимо искать запускаемые файлы. Пути в этом списке разделяются точкой с запятой.
- ❑ Переменная `comspec` указывает имя программы, которая используется в качестве командного процессора.
- ❑ Переменная `OS` содержит информацию о типе операционной системы. Ее значение равно `"Windows_NT"` для всех ОС, построенных по технологии NT (в том числе все ОС, выпущенные после Windows 2000).
- ❑ Переменная `windir` содержит информацию о корневой папке для Windows.
- ❑ Переменная `computername` содержит имя компьютера.
- ❑ Переменная `username` содержит имя пользователя.

Переменные окружения пользователя

Переменные пользователя создаются и модифицируются каждым пользователем. Набор этих переменных специфичен для каждого пользователя. При наличии одинаковых по имени системных переменных и переменных пользователя их значения могут взаимодействовать друг с другом. Так, например, если создана системная переменная `path` и переменная пользователя с этим же именем, то их значения объединяются в одну цепочку.

Локальные переменные окружения

Локальные переменные создаются и модифицируются командой `set`.
Время их жизни ограничено временем работы командного процессора.
Иными словами, если открыто окно команд, то созданные переменные сохраняют свои значения до закрытия этого окна. Переменные, созданные в командном файле, сохраняют свои значения до конца его работы.

Для доступа к переменным любого уровня в командных файлах используется конструкция

%имяпеременной%

например:

```
lh %windir%\system32\kb16 ru
```

Работа с переменными окружения в командных файлах

Локальные переменные создаются и модифицируются командой `set`. Время их жизни ограничено временем работы командного процессора. Иными словами, если открыто окно команд, то созданные переменные сохраняют свои значения до закрытия этого окна. Переменные, созданные в командном файле, сохраняют свои значения до конца его работы.

Для доступа к переменным любого уровня в командных файлах используется конструкция

%имяпеременной%

например:

```
lh %windir%\system32\kb16 ru
```

Работа с переменными окружения в командных файлах (продолжение)

При создании собственных локальных переменных следует иметь в виду, что некоторые конструкции, похожие на доступ к переменным, используются для специальных целей. Это может повлечь двусмысленность. Так, если будет создана переменная `errorlevel`, то выражение `%errorlevel%` возвратит значение этой переменной, а не ожидаемое значение последнего кода завершения!

Локальные переменные затевают системные переменные и переменные пользователя, имеющие такое же имя. Так, если была выполнена команда

```
set path=d:\my_dir
```

то пути, прописанные в переменных более высокого уровня, станут недоступными. Правильный вариант задания такой переменной может быть таким:

```
set path=d:\my_dir;%path%
```

Проверка наличия переменных окружения

Для проверки того, существует ли нужная нам переменная, можно использовать еще одно условие команды if:

```
if [not] defined переменная команда
```

Пример использования этой конструкции:

```
if not defined my_dir (set my_dir=d:\my_dir)
```

Уровни локальных переменных окружения

Локальные переменные могут образовывать несколько уровней. Для чего это надо?

Предположим, Вы написали командный файл, в котором определили свои локальные переменные. Если этот файл будет вызван из другого командного файла, также создающего локальные переменные, в случае совпадения их имен работа вызывающего файла может быть нарушена.

Для предотвращения подобных коллизий следует вынести переменные, создаваемые в каждом командном файле, на свой, более низкий уровень. При совпадении имен переменные более низкого уровня будут затенять переменные, находящиеся на более высоких уровнях. При возврате на более высокий уровень значения переменных, созданных на ниже расположенном уровне, теряются.

Для создания нового уровня локальных переменных служит команда

```
setlocal
```

а для возврата на более высокий уровень – команда

```
endlocal
```

</Отступление 3>



Пример командного файла

Рассмотрим командный файл, выполняющий тестирование олимпиадных решений для районных школьных олимпиад по информатике.

Тестирование состоит из нескольких этапов:

- компиляция с использованием заявленного участником компилятора. Компилятор определяется по расширению файла:

pas	Free Pascal 2.6.2
dpr	Borland Delphi 7.0
cpp	MS Visual Studio 2010
gpp	GNU C++ 4.8.2
g11	GNU C++ 4.8.2 с подключением возможностей стандарта C++11

Пример командного файла (продолжение)

- проверка на наборе тестов. Каждый тест обрабатывается отдельно. При проверке теста возможны ситуации: создан выходной файл; нарушен предел времени; нарушен предел памяти; произошла ошибка выполнения.

В первом случае результаты работы проверяются специальной программой – чекером. В остальных случаях тест считается не пройденным.

Файлы с исходными текстами программ должны находиться в каталоге `source`. Результаты проверки записываются в текстовые файлы с расширением `.res` в каталог `results`.

Пример командного файла (текст батника)

```
@echo off
setlocal

set timer_path=
set cl10_path=
set checker_name=ch_double4
set task_name=Спички - детям не игрушка!
set task_code=matches
set time_limit=1000
set memory_limit=65536
set balls_per_test=2
set input_file=input.txt
set output_file=output.txt
```

Пример командного файла (текст батника, продолжение 1)

```
if %1. ==. goto test_all
call :test0 %1
goto :eof
```

```
:test_all
for %%i in (sources\*.*) do call :test0 %%i
goto :eof
```

```
:test0
```

```
set java=0
if %1. == . goto noparm
if exist results\%~n1.res del results\%~n1.res
echo Task: %task_name% >results\%~n1.res
echo Program to test: %~n1 >>results\%~n1.res
echo ..... >>results\%~n1.res
echo Program to test: %~n1
```

Пример командного файла (текст батника, продолжение 2)

```
if exist sources\%~n1.dpr goto comp_dpr
if exist sources\%~n1.pas goto comp_pas
if exist sources\%~n1.gpp goto comp_gpp
if exist sources\%~n1.g11 goto comp_g11
if exist sources\%~n1.cpp goto comp_cpp
if exist sources\%~n1.java goto comp_java
```

```
:no_source
echo No source file found! >>results\%~n1.res
echo No source file found!
goto :eof
```

Пример командного файла (текст батника, продолжение 3)

```
:comp_java
set java=1
copy sources\%~n1.java . >nul
javac %~n1.java
del %~n1.java >nul
if exist %~n1.class goto testing
goto no_exe
```

```
:comp_dpr
copy sources\%~n1.dpr . >nul
dcc32 -CC -$O+ %~n1.dpr >>results\%~n1.res
:ppc386 -O2 %~n1.pas >>results\%~n1.res
del %~n1.dpr >nul
if exist %~n1.exe goto testing
goto no_exe
```

Пример командного файла (текст батника, продолжение 4)

rem запуск других компиляторов опущен

```
:no_exe
if %java% == 0 (
    echo The executable file isn't found!
    echo The executable file isn't found!
>>results\%~n1.res
) else (
    echo The Java class file isn't found!
    echo The Java class file isn't found!
>>results\%~n1.res
)
goto :eof
```

Пример командного файла (текст батника, продолжение 5)

```
:testing
if exist %~n1.ow del %~n1.ow
if exist %~n1.obj del %~n1.obj
if exist %~n1.o del %~n1.o

for %%i in (*.i) do (
    call :test_one %%~ni %~n1 %task_code%
)

if exist %~n1.exe del %~n1.exe >nul
if exist *.class del *.class >nul
goto :eof
```


Пример командного файла (текст батника, продолжение 6)

```
:noparm
echo Usage: test filename
echo Filename can be with or without extension!
echo The compiler depends on the source file extension:
echo      .dpr          - Borland Delphi 7
echo      .pas          - Free Pascal 2.6.2
echo      .gpp          - MinGV g++
echo      .g11          - MinGV g++ with C++11
echo      .cpp          - Microsoft Visual Studio 2010
echo      .java         - Java SDK (1.5.0 or more)
goto :eof
```

Пример командного файла (текст батника, продолжение 7)

```
:test_one
```

```
@echo off
```

```
@echo Test %1
```

```
@echo *****
```

```
>>results\%2.res
```

```
@echo Test %1 >>results\%2.res
```

```
copy %1.i %input_file% >nul
```

```
if %java% == 0 (
```

```
    %timer_path%timer %2.exe %time_limit% %memory_limit%
```

```
>>results\%2.res
```

```
) else (
```

```
    java %2 2>aaa.aaa
```

```
)
```

```
if errorlevel 1 goto bad_run
```

Пример командного файла (текст батника, продолжение 8)

```
if exist checker.in del checker.in
echo 1 > checker.in
echo %input_file% >> checker.in
echo 1 >> checker.in
echo %1.o >> checker.in
echo %output_file% >> checker.in
echo 999 >> checker.in
echo %3 >> checker.in
echo %1 >> checker.in
echo %balls_per_test% >> checker.in
echo 1 >> checker.in
%checker_name%
copy results\%2.res+checker.out results\%2.res >nul
goto clean
```

Пример командного файла (текст батника, продолжение 9)

```
:bad_run
echo Everything fail!
:clean
if exist checker.in del checker.in
if exist checker.out del checker.out
if exist %input_file% del %input_file%
if exist %output_file% del %output_file%
if exist aaa.aaa del aaa.aaa

:eof
endlocal
```