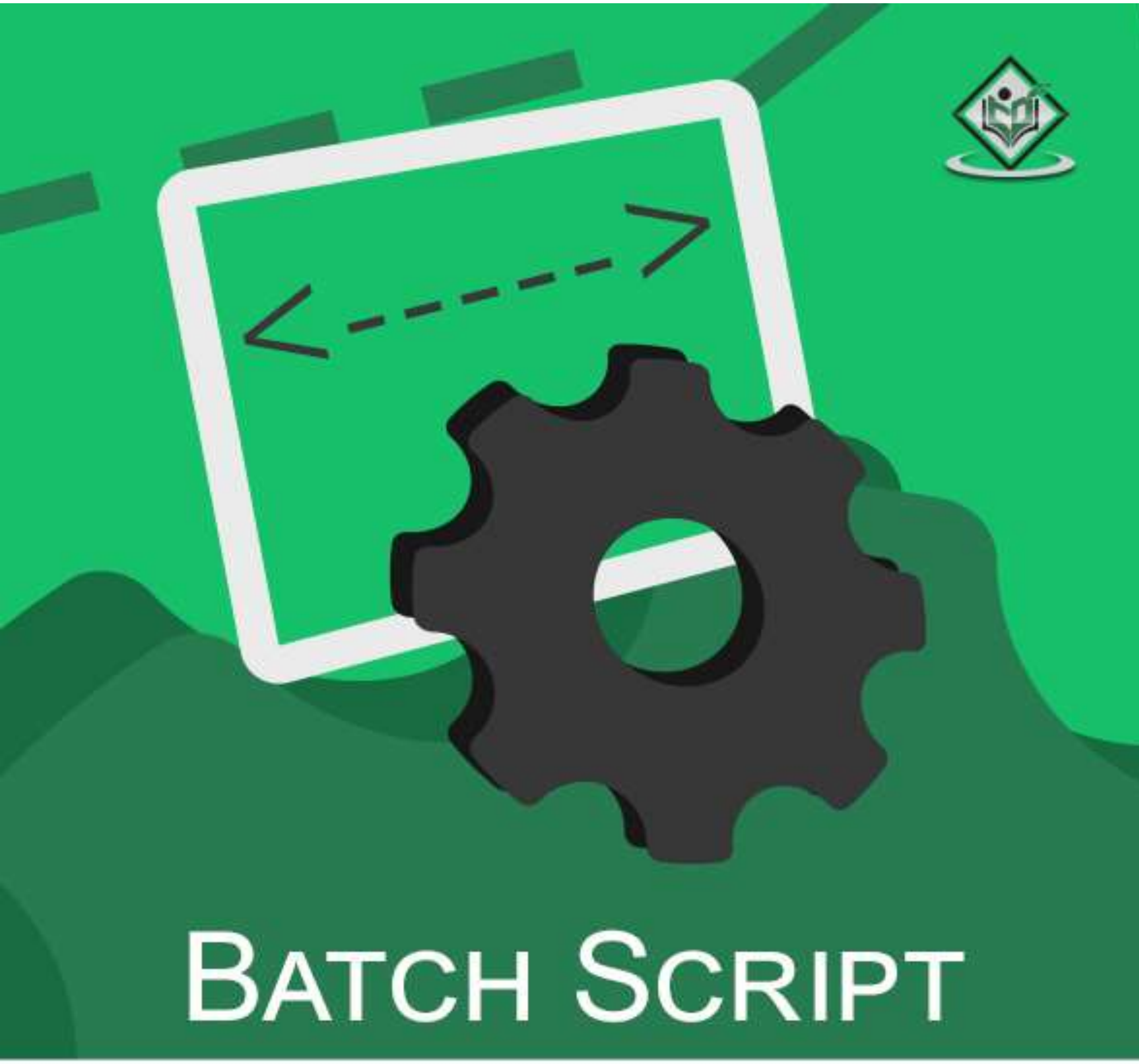


Batch Script

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About this Tutorial

Batch Scripts are stored in simple text files containing lines with commands that get executed in sequence, one after the other. Scripting is a way by which one can alleviate this necessity by automating these command sequences in order to make one's life at the shell easier and more productive.

This tutorial discusses the basic functionalities of Batch Script along with relevant examples for easy understanding.

Audience

This tutorial has been prepared for beginners to understand the basic concepts of Batch Script.

Prerequisites

A reasonable knowledge of computer programming and concepts such as variables, commands, syntax, etc. is desired.

Copyright & Disclaimer

Table of Contents

1. Batch Script – Overview

Batch Script is incorporated to automate command sequences which are repetitive in nature. Scripting is a way by which one can alleviate this necessity by automating these command sequences in order to make one’s life at the shell easier and more productive. In most organizations, Batch Script is incorporated in some way or the other to automate stuff.

Some of the features of Batch Script are:

- Can read inputs from users so that it can be processed further.
- Has control structures such as for, if, while, switch for better automating and scripting.
- Supports advanced features such as Functions and Arrays.
- Supports regular expressions.
- Can include other programming codes such as Perl.

Some of the common uses of Batch Script are:

- Setting up servers for different purposes.
- Automating housekeeping activities such as deleting unwanted files or log files.
- Automating the deployment of applications from one environment to another.
- Installing programs on various machines at once.

Batch scripts are stored in simple text files containing lines with commands that get executed in sequence, one after the other. These files have the special extension BAT or CMD. Files of this type are recognized and executed through an interface (sometimes called a shell) provided by a system file called the command interpreter. On Windows systems, this interpreter is known as cmd.exe.

Running a batch file is a simple matter of just clicking on it. Batch files can also be run in a command prompt or the Start-Run line. In such case, the full path name must be used unless the file's path is in the path environment. Following is a simple example of a batch script. This batch script when run deletes all files in the current directory.

```
:: Deletes All files in the Current Directory With Prompts and Warnings
::(Hidden, System, and Read-Only Files are Not Affected)
:: @ECHO OFF
DEL . DR
```

2. Batch Script – Environment

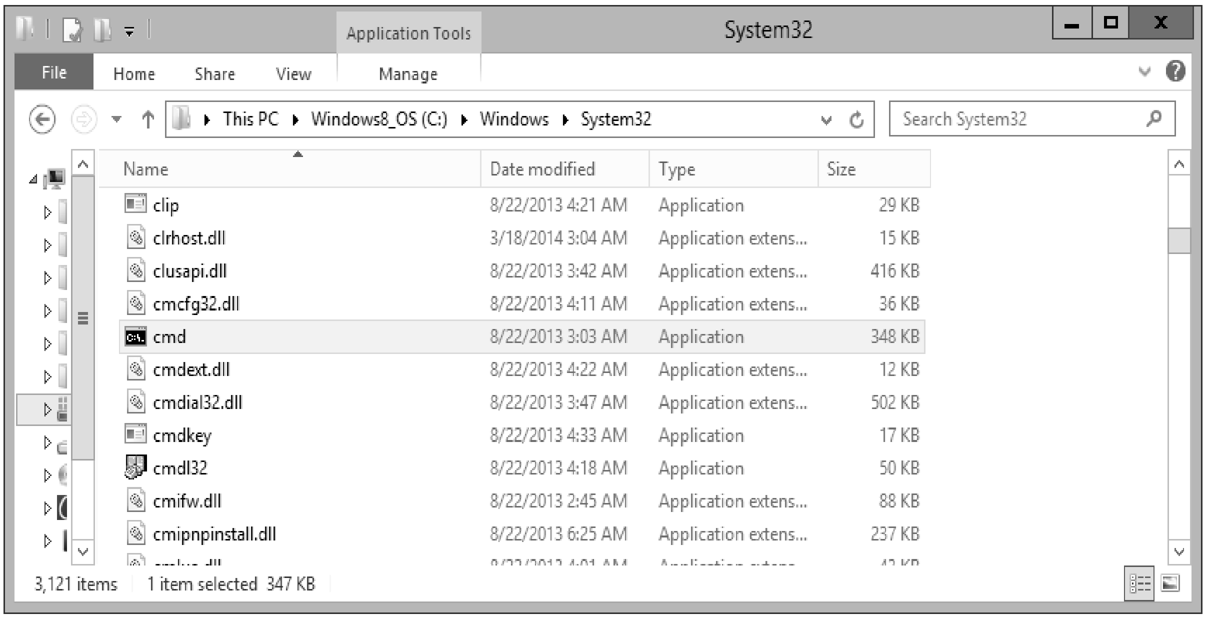
This chapter explains the environment related to Batch Script.

Writing and Executing

Typically, to create a batch file, notepad is used. This is the simplest tool for creation of batch files. Next is the execution environment for the batch scripts. On Windows systems, this is done via the command prompt or cmd.exe. All batch files are run in this environment.

Following are the different ways to launch cmd.exe:

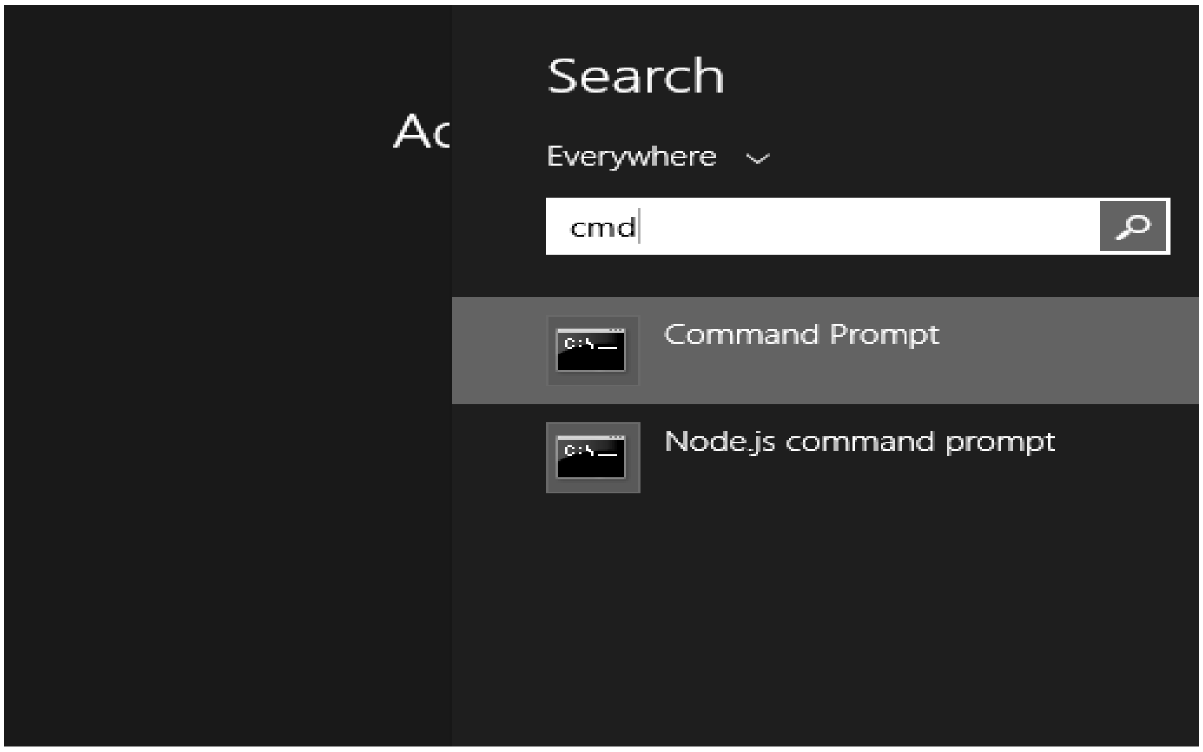
Method 1: Go to C:\Windows\System32 and double click on the cmd file.



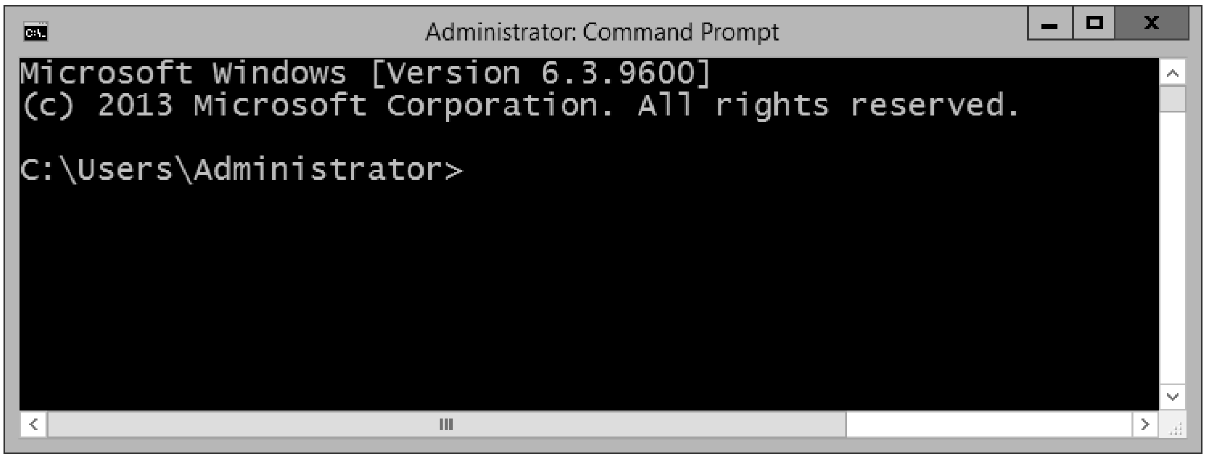
Name	Date modified	Type	Size
clip	8/22/2013 4:21 AM	Application	29 KB
clrhost.dll	3/18/2014 3:04 AM	Application extens...	15 KB
clusapi.dll	8/22/2013 3:42 AM	Application extens...	416 KB
cmcfg32.dll	8/22/2013 4:11 AM	Application extens...	36 KB
cmd	8/22/2013 3:03 AM	Application	348 KB
cmdext.dll	8/22/2013 4:22 AM	Application extens...	12 KB
cmdial32.dll	8/22/2013 3:47 AM	Application extens...	502 KB
cmdkey	8/22/2013 4:33 AM	Application	17 KB
cmdl32	8/22/2013 4:18 AM	Application	50 KB
cmifw.dll	8/22/2013 2:45 AM	Application extens...	88 KB
cmipnpinstall.dll	8/22/2013 6:25 AM	Application extens...	237 KB

3,121 items 1 item selected 347 KB

Method 2: Via the run command – The following snapshot shows to find the command prompt(cmd.exe) on Windows server 2012.



Once the cmd.exe is launched, you will be presented with the following screen. This will be your environment for executing your batch scripts.



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>
```

Environment Variables

In order to run batch files from the command prompt, you either need to go to the location to where the batch file is stored or alternatively you can enter the file location in the path environment variable. Thus assuming that the batch file is stored in the location C:\Application\bin, you would need to follow these instructions for the PATH variable inclusion.

OS	Output
Windows	Append the String; C:\Application\bin to the end of the system variable PATH.

3. Batch Script – Commands

In this chapter, we will look at some of the frequently used batch commands.

ver

This batch command shows the version of MS-DOS you are using.

Syntax

```
ver
```

Example

```
@echo off
ver
```

Output

The output of the above command is as follows. The version number will depend upon the operating system you are working on.

```
Microsoft Windows [Version 6.3.9600]
```

ASSOC

This is a batch command that associates an extension with a file type (FTYPE), displays existing associations, or deletes an association.

Syntax

```
assoc – Displays all the file extensions
assoc | find “.ext” – Displays only those file extensions which have the
extension ext.
```

Example

```
@echo off
assoc > C:\lists.txt
assoc | find “.doc” > C:\listsdoc.txt
```

Output

The list of file associations will be routed to the file lists.txt. The following output shows what is there in the listsdoc.txt file after the above batch file is run.

```
.doc=Word.Document.8
.dochtml=wordhtmlfile
.docm=Word.DocumentMacroEnabled.12
.docmhtml=wordmhtmlfile
.docx=Word.Document.12
.docxml=wordxmlfile
```

CD

This batch command helps in making changes to a different directory, or displays the current directory.

Syntax

```
cd
```

Example

The following example shows how the cd command can be used in a variety of ways.

```
@echo off
Rem The cd without any parameters is used to display the current working
directory
cd
Rem Changing the path to Program Files
cd\Program Files
cd
Rem Changing the path to Program Files
cd %USERPROFILE%
cd
Rem Changing to the parent directory
cd..
cd
Rem Changing to the parent directory two levels up
cd..\..
cd
```

Output

The above command will display the following output after changing to the various folder locations.

```
C:\Users\Administrator
C:\Program Files
C:\Users\Administrator
C:\Users
C:\
```

CLS

This batch command clears the screen.

Syntax

```
cls
```

Example

```
@echo off
Cls
```

Output

The command prompt screen will be cleared.

Copy

This batch command is used for copying files from one location to the other.

Syntax

```
Copy [source] [destination]
```

The files will be copied from source to destination location.

Example

The following example shows the different variants of the **copy** command.

```
@echo off
cd
Rem Copies lists.txt to the present working directory. If there is no
destination identified , it defaults to the present working directory.
copy c:\lists.txt
```

```
Rem The file lists.txt will be copied from C:\ to C:\tp location
copy C:\lists.txt c:\tp
Rem Quotation marks are required if the file name contains spaces
copy “C:\My File.txt”
Rem Copies all the files in F drive which have the txt file extension to the
current working directory
copy F:\*.txt
Rem Copies all files from dirA to dirB. Note that directories nested in dirA
will not be copied
copy C:\dirA dirB
```

Output

All actions are performed as per the remarks in the batch file.

DEL

This batch command deletes files and not directories.

Syntax

```
del [filename]
```

Example

The following example shows the different variants of the **del** command.

```
@echo off
Rem Deletes the file lists.txt in C:\
del C:\lists.txt
Rem Deletes all files recursively in all nested directories
del /s *.txt
Rem Deletes all files recursively in all nested directories , but asks for the
confirmation from the user first
Del /p /s *.txt
```

Output

All actions are performed as per the remarks in the batch file.

DIR

This batch command lists the contents of a directory.

Syntax

```
dir
```

Example

The following example shows the different variants of the **dir** command.

```
@echo off

Rem All the directory listings from C:\ will be routed to the file lists.txt

dir C:\>C:\lists.txt

Rem Lists all directories and subdirectories recursively

dir /s

Rem Lists the contents of the directory and all subdirectories recursively, one
file per line, displaying complete path for each listed file or directory.

dir /s /b

Rem Lists all files with .txt extension.

dir *.txt

Rem Includes hidden files and system files in the listing.

dir /a

Rem Lists hidden files only.

dir /ah
```

Output

All actions are performed as per the remarks in the batch file.

DATE

This batch command help to find the system date.

Syntax

```
DATE
```

Example

```
@echo off

echo %DATE%
```

Output

The current date will be displayed in the command prompt. For example,

```
Mon 12/28/2015
```

ECHO

This batch command displays messages, or turns command echoing on or off.

Syntax

```
ECHO “string”
```

Example

The following example shows the different variants of the dir command.

```
Rem Turns the echo on so that each command will be shown as executed
echo on
echo "Hello World"
Rem Turns the echo off so that each command will not be shown when executed
@echo off
echo "Hello World"
Rem Displays the contents of the PATH variable
echo %PATH%
```

Output

The following output will be displayed in the command prompt.

```
C:\>Rem Turns the echo on so that each command will be shown as executed

C:\>echo on

C:\>echo "Hello World"
"Hello World"

C:\>Rem Turns the echo off so that each command will not be shown when executed

"Hello World"
C:\Users\ADMINI~1\AppData\Local\Temp
```

EXIT

This batch command exits the DOS console.

Syntax

```
Exit
```

Example

```
@echo off
echo "Hello World"
exit
```

Output

The batch file will terminate and the command prompt window will close.

MD

This batch command creates a new directory in the current location.

Syntax

```
md [new directory name]
```

Example

```
@echo off
md newdir
cd newdir
cd
Rem “Goes back to the parent directory and create 2 directories”
cd..
md newdir1 newdir1
cd newdir1
cd
cd..
cd newdir2
cd
```

Output

The above command produces the following output.

```
C:\newdir
C:\newdir1
C:\newdir2
```

MOVE

This batch command moves files or directories between directories.

Syntax

```
move [source] [destination]
```

The files will be copied from source to destination location.

Example

The following example shows the different variants of the move command.

```
@echo off
Rem Moves the file list.txt to the directory c:\tp
move C:\lists.txt c:\tp
Rem Renames directory Dir1 to Dir2, assuming Dir1 is a directory and Dir2 does
not exist.
move Dir1 Dir2
Rem Moves the file lists.txt to the current directory.
move C:\lists.txt
```

Output

All actions are performed as per the remarks in the batch file.

PATH

This batch command displays or sets the path variable.

Syntax

```
PATH
```

Example

```
@echo off
Echo %PATH%
```

Output

The value of the path variable will be displayed in the command prompt.

PAUSE

This batch command prompts the user and waits for a line of input to be entered.

Syntax

```
Pause
```

Example

```
@echo off
pause
```

Output

The command prompt will show the message "Press any key to continue…." to the user and wait for the user's input.

PROMPT

This batch command can be used to change or reset the **cmd.exe** prompt.

Syntax

```
PROMPT [newpromptname]
```

Example

```
@echo off
prompt myprompt$G
```

The $G is the greater than sign which is added at the end of the prompt.

Output

The prompt shown to the user will now be **myprompt>**

RD

This batch command removes directories, but the directories need to be empty before they can be removed.

Syntax

```
rd [directoryname]
```

Example

The following example shows the different variants of the **rd** command.

```
@echo off
Rem removes the directory called newdir
rd C:\newdir
Rem removes 2 directories
rd Dir1 Dir2
Rem Removes directory with spaces
rd "Application A"
Rem Removes the directory Dir1 including all the files and subdirectories in it
rd /s Dir1
Rem Removes the directory Dir1 including all the files and subdirectories in it
but asks for a user confirmation first.
rd /q /s Dir1
```

Output

All actions are performed as per the remarks in the batch file.

REN

Renames files and directories.

Syntax

```
ren [oldfile/dirname] [newfile/dirname]
```

Renames the file name from the old file/dir name to the new one.

Example

```
@echo off
ren C:\lists.txt C:\newlists.txt
```

Output

The file **lists.txt** will be renamed to **newlists.txt**.

REM

This batch command is used for remarks in batch files, preventing the content of the remark from being executed.

Syntax

```
REM remark description
```

Example

```
@echo off
REM This is a batch file
```

Output

```
None
```

START

This batch command starts a program in new window, or opens a document.

Syntax

```
START “programname”
```

Example

```
@echo off
start notepad.exe
```

Output

When the batch file is executed, a new notepad windows will start.

TIME

This batch command sets or displays the time.

Syntax

```
TIME
```

Example

```
@echo off
echo %TIME%
```

Output

The current system time will be displayed. For example,

```
22:06:52.87
```

TYPE

This batch command prints the content of a file or files to the output.

Syntax

```
TYPE [filename]
```

Where filename is the file whose contents need to be displayed.

Example

```
@echo off
TYPE C:\tp\lists.txt
```

Output

The contents of the file lists.txt will be displayed to the command prompt.

VOL

This batch command displays the volume labels.

Syntax

```
VOL
```

Example

```
@echo off
VOL
```

Output

The output will display the current volume label. For example,

```
Volume in drive C is Windows8_OS
 Volume Serial Number is E41C-6F43
```

ATTRIB

Displays or sets the attributes of the files in the current directory.

Syntax

```
attrib
```

Example

The following example shows the different variants of the attrib command.

```
@echo off
Rem Displays the attribites of the file in the current directory
Attrib
Rem Displays the attributes of the file lists.txt
attrib C:\tp\lists.txt
Rem Adds the "Read-only" attribute to the file.
attrib +r C:\tp\lists.txt
Attrib C:\tp\lists.txt
Rem Removes the "Archived" attribute from the file
attrib -a C:\tp\lists.txt
Attrib C:\tp\lists.txt
```

Output

For example,

```

A            C:\tp\assoclst.txt
A            C:\tp\List.cmd
A            C:\tp\lists.txt
A            C:\tp\listsA.txt
A            C:\tp\lists.txt
A    R       C:\tp\lists.txt
     R       C:\tp\lists.txt
```

CHKDSK

This batch command checks the disk for any problems.

Syntax

```
chkdsk
```

Example

```
@echo off
chkdsk
```

Output

The above command starts checking the current disk for any errors.

CHOICE

This batch command provides a list of options to the user.

Syntax

```
CHOICE /c [Options] /m [Message]
```

Where Options is the list of options to provide to the user and Message is the string message which needs to be displayed.

Example

```
@echo off
echo "What is the file size you what"
echo "A:10MB"
echo "B:20MB"
echo "C:30MB"
choice /c ABC /m "What is your option A , B or C"
```

Output

The above program produces the following output.

```
"What is the file size you what"
"A:10MB"
"B:20MB"
"C:30MB"
```

```
What is your option A , B or C [A,B,C]?
```

CMD

This batch command invokes another instance of command prompt.

Syntax

```
cmd
```

Example

```
@echo off
cmd
```

Output

Another instance of command prompt will be invoked.

COMP

This batch command compares 2 files based on the file size.

Syntax

```
COMP [sourceA] [sourceB]
```

Wherein sourceA and sourceB are the files which need to be compared.

Example

```
@echo off
COMP C:\tp\lists.txt C:\tp\listsA.txt
```

Output

The above command will compare the files lists.txt and listsA.txt and find out if the two file sizes are different.

CONVERT

This batch command converts a volume from FAT16 or FAT32 file system to NTFS file system.

Syntax

```
CONVERT [drive]
```

Example

```
@echo off
CONVERT C:\
```

Output

The above command will convert the file system of C drive.

DRIVERQUERY

This batch command shows all installed device drivers and their properties.

Syntax

```
driverquery
```

Example

```
@echo off
driverquery
```

Output

The above command will display the information of all the device drivers installed on the current system. Following is an example of a subset of the information displayed.

```
WacomPen     Wacom Serial Pen HID D Kernel        8/22/2013 4:39:15 AM
Wanarp       Remote Access IP ARP D Kernel        8/22/2013 4:35:45 AM
Wanarpv6     Remote Access IPv6 ARP Kernel        8/22/2013 4:35:45 AM
Wdf01000     Kernel Mode Driver Fra Kernel        8/22/2013 4:38:56 AM
WFPLWFS      Microsoft Windows Filt Kernel        11/9/2014 6:57:28 PM
WIMMount     WIMMount               File System   8/22/2013 4:39:34 AM
WinMad       WinMad Service         Kernel        5/9/2013 9:14:27 AM
WinNat       Windows NAT Driver     Kernel        1/22/2014 1:10:49 AM
WinUsb       WinUsb Driver          Kernel        8/22/2013 4:37:55 AM
WinVerbs     WinVerbs Service       Kernel        5/9/2013 9:14:30 AM
WmiAcpi      Microsoft Windows Mana Kernel        8/22/2013 4:40:04 AM
WpdUpFltr    WPD Upper Class Filter Kernel        8/22/2013 4:38:45 AM
ws2ifsl      Windows Socket 2.0 Non Kernel        8/22/2013 4:40:03 AM
```

```
wtlmdrv      Microsoft iSCSI Target Kernel        8/22/2013 4:39:19 AM
WudfPf       User Mode Driver Frame Kernel        8/22/2013 4:37:21 AM
WUDFWpdFs    WUDFWpdFs              Kernel        8/22/2013 4:36:50 AM
WUDFWpdMtp   WUDFWpdMtp             Kernel        8/22/2013 4:36:50 AM
```

EXPAND

This batch command extracts files from compressed .cab cabinet files.

Syntax

```
EXPAND [cabinetfilename]
```

Example

```
@echo off
EXPAND excel.cab
```

Output

The above command will extract the contents of the file excel.cab in the current location.

FIND

This batch command searches for a string in files or input, outputting matching lines.

Syntax

```
FIND [text] [destination]
```

Where text is the string which needs to be searched for and destination is the source in which the search needs to take place.

Example

```
@echo off
FIND "Application" C:\tp\lists.txt
```

Output

If the word "Application" resides in the file lists.txt, the line containing the string will be displayed in the command prompt.

FORMAT

This batch command formats a disk to use Windows-supported file system such as FAT, FAT32 or NTFS, thereby overwriting the previous content of the disk.

Syntax

```
format [drive]
```

Where drive is the drive which needs to be formatted.

Example

```
@echo off
format D:\
```

Output

With the above command, D drive will be formatted.

HELP

This batch command shows the list of Windows-supplied commands.

Syntax

```
help
```

Example

```
@echo off
help
```

Output

The above command will display a list of all commands and their description. Following is an example of a subset of the output.

```
SCHTASKS      Schedules commands and programs to run on a computer.
SHIFT         Shifts the position of replaceable parameters in batch files.
SHUTDOWN      Allows proper local or remote shutdown of machine.
SORT          Sorts input.
START         Starts a separate window to run a specified program or command.
SUBST         Associates a path with a drive letter.
SYSTEMINFO    Displays machine specific properties and configuration.
TASKLIST      Displays all currently running tasks including services.
TASKKILL      Kill or stop a running process or application.
TIME          Displays or sets the system time.
TITLE         Sets the window title for a CMD.EXE session.
TREE          Graphically displays the directory structure of a drive or
```

```
               path.
TYPE           Displays the contents of a text file.
VER            Displays the Windows version.
VERIFY         Tells Windows whether to verify that your files are written
               correctly to a disk.
VOL            Displays a disk volume label and serial number.
XCOPY          Copies files and directory trees.
WMIC           Displays WMI information inside interactive command shell.

For more information on tools see the command-line reference in the online
help.
```

IPCONFIG

This batch command displays Windows IP Configuration. Shows configuration by connection and the name of that connection.

Syntax

```
ipconfig
```

Example

```
@echo off
ipconfig
```

Output

The above command will display the Windows IP configuration on the current machine. Following is an example of the output.

```
Windows IP Configuration



Wireless LAN adapter Local Area Connection* 11:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Ethernet adapter Ethernet:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
```

```

Wireless LAN adapter Wi-Fi:
   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Tunnel adapter Teredo Tunneling Pseudo-Interface:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
```

LABEL

This batch command adds, sets or removes a disk label.

Syntax

```
Label
```

Example

```
@echo off
label
```

Output

The above command will prompt the user to enter a new label for the current drive.

MORE

This batch command displays the contents of a file or files, one screen at a time.

Syntax

```
More [filename]
```

Where filename is the file whose contents need to be listed one screen at a time.

Example

```
@echo off
More C:\tp\lists.txt
Directory of C:\Program Files
```

Output

The above command will display the contents of the file lists.txt one screen at a time. Following is an example of an output. Note the -- More (12%) – at the end of the screen. In order to proceed and display the remaining contents of the file, you need to enter a key.

```
12/22/2015  02:31 AM    <DIR>          .
12/22/2015  02:31 AM    <DIR>          ..
12/15/2015  11:14 PM    <DIR>          Application Verifier
12/18/2015  05:06 PM    <DIR>          Bonjour
12/26/2015  08:01 PM    <DIR>          CCleaner
12/18/2015  05:05 PM    <DIR>          Common Files
12/17/2015  11:04 AM    <DIR>          Git
12/15/2015  11:09 PM    <DIR>          IIS
12/15/2015  11:10 PM    <DIR>          IIS Express
12/15/2015  10:16 PM    <DIR>          Intel
03/18/2014  02:24 AM    <DIR>          Internet Explorer
12/18/2015  05:06 PM    <DIR>          iPod
12/18/2015  05:06 PM    <DIR>          iTunes
12/15/2015  11:16 PM    <DIR>          Microsoft Identity Extensions
12/15/2015  11:46 PM    <DIR>          Microsoft Office
12/22/2015  02:31 AM    <DIR>          Microsoft Silverlight
12/15/2015  11:15 PM    <DIR>          Microsoft SQL Server
12/15/2015  11:15 PM    <DIR>          Microsoft SQL Server Compact Edition
12/15/2015  10:59 PM    <DIR>          Microsoft Visual Studio 12.0
-- More (12%) --
```

NET

Provides various network services, depending on the command used.

Syntax

```
NET [variant]
```

Where its variants can be one of the following:

- net accounts
- net computer
- net config
- net continue
- net file
- net group
- net help
- net helpmsg
- net localgroup
- net name
- net pause
- net print
- net send
- net session
- net share
- net start
- net statistics
- net stop
- net time
- net use
- net user
- net view

Example

```
@echo off
Net user
```

Output

The above command will display the current accounts defined on the system. Following is an example of an output.

```
User accounts for \\WIN-50GP30FGO75


-------------------------------------------------------------------------------
Administrator            atlbitbucket             Guest
The command completed successfully.
```

PING

This batch command sends ICMP/IP "echo" packets over the network to the designated address.

Syntax

```
PING [address]
```

Where address is the IP address or hostname of the destination system.

Example

```
@echo off
Ping 127.0.0.1
```

Output

The above command will send ICMP/IP "echo" packets to the destination address 192.168.0.1. Following is an example of the output.

```
Pinging 127.0.0.1 with 32 bytes of data:
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128

Ping statistics for 127.0.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

SHUTDOWN

This batch command shuts down a computer, or logs off the current user.

Syntax

```
shutdown
```

Example

```
@echo off
shutdown
```

Output

If the user executing the batch files has the relevant rights, the computer will be shutdown.

SORT

This batch command takes the input from a source file and sorts its contents alphabetically, from A to Z or Z to A. It prints the output on the console.

Syntax

```
Sort [filename]
```

Where filename is the file whose contents need to be sorted.

Example

```
@echo off
Sort C:\tp\lists.txt
```

SUBST

This batch command assigns a drive letter to a local folder, displays current assignments, or removes an assignment.

Syntax

```
Subst [driveletter]
```

Example

```
@echo off
Subst p:
```

Output

P: will be assigned as the drive letter for the current folder.

SYSTEMINFO

This batch command shows configuration of a computer and its operating system.

Syntax

```
systeminfo
```

Example

```
@echo off
systeminfo
```

Output

The above command will show the system information on the current system. Following is a subset of the output.

```
Host Name:                 WIN-50GP30FGO75
OS Name:                   Microsoft Windows Server 2012 R2 Standard
OS Version:                6.3.9600 N/A Build 9600
OS Manufacturer:           Microsoft Corporation
OS Configuration:          Standalone Server
OS Build Type:             Multiprocessor Free
Registered Owner:          Windows User
Registered Organization:
Product ID:                00252-70000-00000-AA535
Original Install Date:     12/13/2015, 12:10:16 AM
System Boot Time:          12/28/2015, 4:43:04 PM
System Manufacturer:       LENOVO
System Model:              20287
System Type:               x64-based PC
```

TASKKILL

This batch command ends one or more tasks.

Syntax

```
Taskkill /im [taskname]
```

Example

```
@echo off
Taskkill /im mspaint.exe
```

Output

The above command will send a termination message to any open programs of MS Paint.

TASKLIST

This batch command lists tasks, including task name and process id (PID).

Syntax

```
Tasklist
```

Example

```
@echo off
Tasklist
```

Output

The above command will list all the tasks on the current system.

XCOPY

This batch command copies files and directories in a more advanced way.

Syntax

```
Xcopy [source][destination]
```

Example

```
Xcopy c:\lists.txt c:\tp\
```

Output

The above command will copy the file lists.txt to the tp folder.

TREE

This batch command displays a tree of all subdirectories of the current directory to any level of recursion or depth.

Syntax

```
Tree
```

Example

```
@echo off
tree
```

Output

The above command will display the tree structure of the current directory. Following is an example of the output.

```
Folder PATH listing for volume Windows8_OS
Volume serial number is E41C-6F43
C:.
 ├───────newdir
 ├───────newdir1
 └───────newdir2
```

FC

This batch command lists the actual differences between two files.

Syntax

```
Fc [fileA] [fileB]
```

Where fileA and fileB are 2 files that need to be compared.

Example

```
@echo off
FC lists.txt listsA.txt
```

Output

The above command will display the differences in the contents of the files (lists.txt and listsA.txt) if any.

DISKPART

This batch command shows and configures the properties of disk partitions.

Syntax

```
Diskpart
```

Example

```
@echo off
diskpart
```

Output

The above command shows the properties of disk partitions. Following is an example of the output.

```
Microsoft DiskPart version 6.3.9600


Copyright (C) 1999-2013 Microsoft Corporation.
On computer: WIN-50GP30FGO75
```

TITLE

This batch command sets the title displayed in the console window.

Syntax

```
TITLE [Tilename]
```

Where tilename is the new name to be given to the title of the command prompt window.

Example

```
@echo off
Title “New Windows Title”
```

Output

The above command will change the title of the window to “New Windows Title”.

SET

Displays the list of environment variables on the current system.

Syntax

```
Set
```

Example

```
@echo off
set
```

Output

The above command displays the list of environment variables on the current system.

4. Batch Script – Files

In this chapter, we will learn how to create, save, execute, and modify batch files.

Creating Batch Files

Batch files are normally created in notepad. Hence the simplest way is to open notepad and enter the commands required for the script. For this exercise, open notepad and enter the following statements.

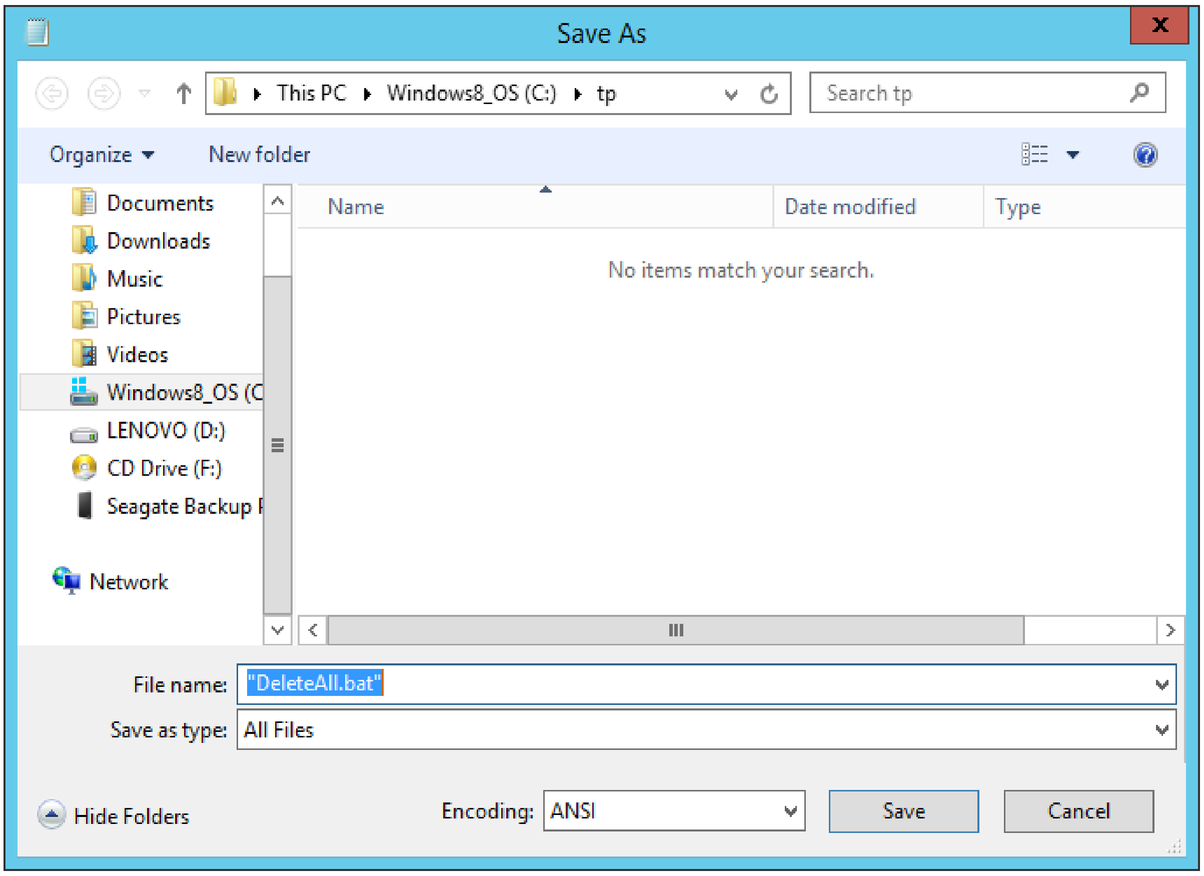
```
:: Deletes All files in the Current Directory With Prompts and Warnings
::(Hidden, System, and Read-Only Files are Not Affected)
::
@ECHO OFF
DEL .
DR
```

Saving Batch Files

After your batch file is created, the next step is to save your batch file. Batch files have the extension of either .bat or .cmd. Some general rules to keep in mind when naming batch files:

- Try to avoid spaces when naming batch files, it sometime creates issues when they are called from other scripts.

- Don't name them after common batch files which are available in the system such as ping.cmd.



No items match your search.

File name: "DeleteAll.bat"

Save as type: All Files

Encoding: ANSI

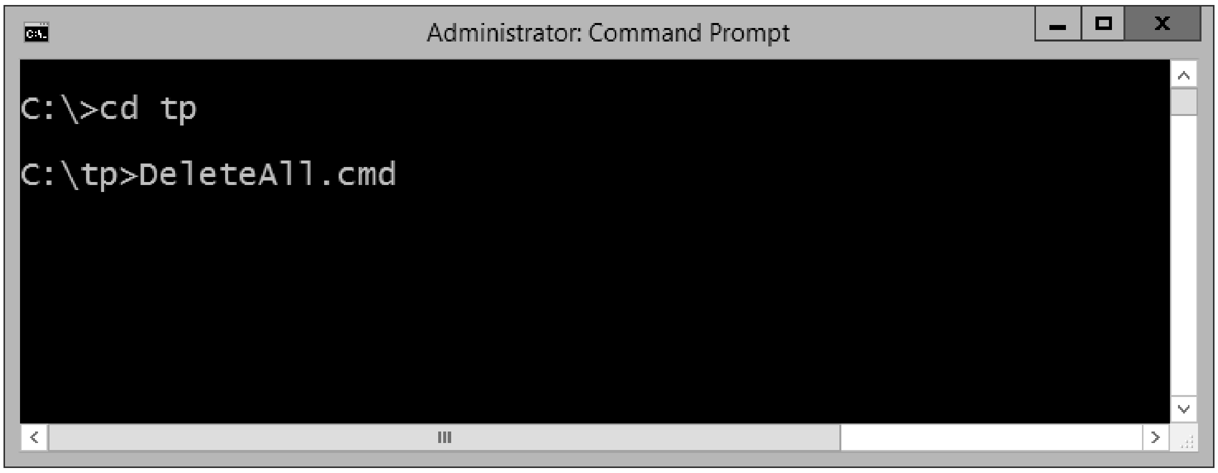
The above screenshot shows how to save the batch file. When saving your batch file a few points to keep in mind.

- Remember to put the .bat or .cmd at the end of the file name.
- Choose the "Save as type" option as "All Files".
- Put the entire file name in quotes "".

Executing Batch Files

Following are the steps to execute a batch file:

- **Step 1**: Open the command prompt (cmd.exe).
- **Step 2**: Go to the location where the .bat or .cmd file is stored.
- **Step 3**: Write the name of the file as shown in the following image and press the Enter button to execute the batch file.



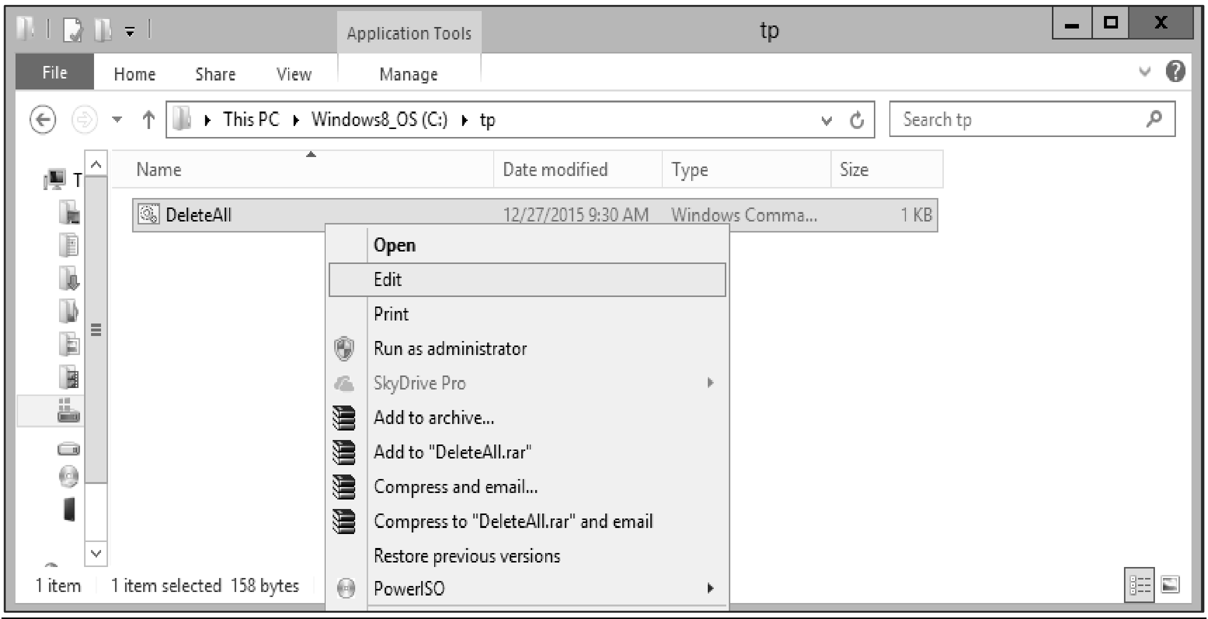
```
C:\>cd tp

C:\tp>DeleteAll.cmd
```

Modifying Batch Files

Following are the steps for modifying an existing batch file.

- **Step 1**: Open windows explorer.
- **Step 2**: Go to the location where the .bat or .cmd file is stored.
- **Step 3**: Right-click the file and choose the "Edit" option from the context menu. The file will open in Notepad for further editing.



5. Batch Script – Syntax

Normally, the first line in a batch file often consists of the following command.

ECHO Command

```
@echo off
```

By default, a batch file will display its command as it runs. The purpose of this first command is to turn off this display. The command "echo off" turns off the display for the whole script, except for the "echo off" command itself. The "at" sign "@" in front makes the command apply to itself as well.

Documentation

Very often batch files also contains lines that start with the "Rem" command. This is a way to enter comments and documentation. The computer ignores anything on a line following Rem. For batch files with increasing amount of complexity, this is often a good idea to have comments.

First Batch Script Program

Let's construct our simple first batch script program. Open notepad and enter the following lines of code. Save the file as "List.cmd".

The code does the following:

- Uses the echo off command to ensure that the commands are not shown when the code is executed.
- The Rem command is used to add a comment to say what exactly this batch file does.
- The dir command is used to take the contents of the location C:\Program Files.
- The '>' command is used to redirect the output to the file C:\lists.txt.
- Finally, the echo command is used to tell the user that the operation is completed.

```
@echo off
Rem This is for listing down all the files in the directory Program files
dir "C:\Program Files" > C:\lists.txt
echo "The program has completed"
```

When the above command is executed, the names of the files in C:\Program Files will be sent to the file C:\Lists.txt and in the command prompt the message "The program has completed" will be displayed.

6. Batch Script – Variables

There are two types of variables in batch files. One is for parameters which can be passed when the batch file is called and the other is done via the set command.

Command Line Arguments

Batch Script supports the concept of command line arguments wherein arguments can be passed to the batch file when invoked. The arguments can be called from the batch files through the variables %1, %2, %3, and so on.

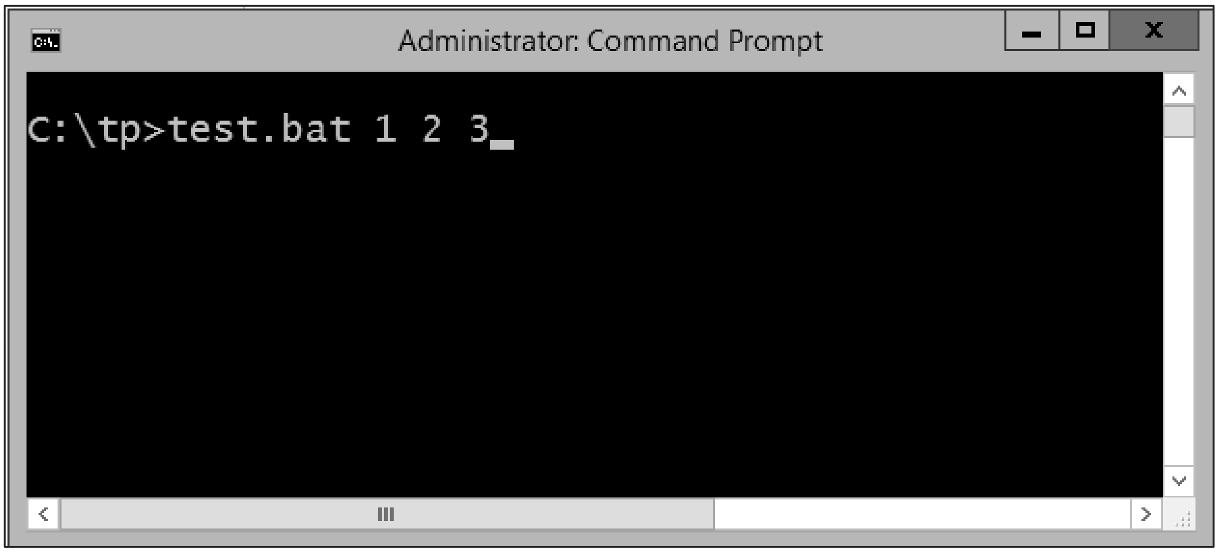
The following example shows a batch file which accepts 3 command line arguments and echo's them to the command line screen.

```
@echo off
echo %1
echo %2
echo %3
```

If the above batch script is stored in a file called test.bat and we were to run the batch as

```
Test.bat 1 2 3
```

Following is a screenshot of how this would look in the command prompt when the batch file is executed.



```
C:\tp>test.bat 1 2 3_
```

The above command produces the following output.

```
1
2
3
```

If we were to run the batch as

```
Example 1 2 3 4
```

The output would still remain the same as above. However, the fourth parameter would be ignored.

Set Command

The other way in which variables can be initialized is via the 'set' command. Following is the syntax of the set command.

Syntax

```
set /A variable-name = value
```

where,

- **variable-name** is the name of the variable you want to set.
- **value** is the value which needs to be set against the variable.
- **/A –** This switch is used if the value needs to be numeric in nature.

The following example shows a simple way the set command can be used.

Example

```
@echo off
set message=Hello World
echo %message%
```

- In the above code snippet, a variable called message is defined and set with the value of "Hello World".
- To display the value of the variable, note that the variable needs to be enclosed in the % sign.

Output

The above command produces the following output.

```
Hello World
```

Working with Numeric Values

In Batch Script, it is also possible to define a variable to hold a numeric value. This can be done by using the /A switch.

The following code shows a simple way in which numeric values can be set with the /A switch.

```
@echo off
SET /A a=5
SET /A b=10
SET /A c=%a% + %b%
echo %c%
```

- We are first setting the value of 2 variables, a and b to 5 and 10 respectively.
- We are adding those values and storing in the variable c.
- Finally, we are displaying the value of the variable c.

The output of the above program would be 15.

All of the arithmetic operators work in batch files. The following example shows arithmetic operators can be used in batch files.

```
@echo off
SET /A a=5
SET /A b=10
SET /A c=%a% + %b%
echo %c%
SET /A c=%a% - %b%
echo %c%
SET /A c=%b% / %a%
echo %c%
SET /A c=%b% * %a%
echo %c%
```

The above command produces the following output.

```
15
-5
2
20
```

Local vs Global Variables

In any programming language, there is an option to mark variables as having some sort of scope, i.e. the section of code on which they can be accessed. Normally, variable having a global scope can be accessed anywhere from a program whereas local scoped variables have a defined boundary in which they can be accessed.

DOS scripting also has a definition for locally and globally scoped variables. By default, variables are global to your entire command prompt session. Call the SETLOCAL command to make variables local to the scope of your script. After calling SETLOCAL, any variable assignments revert upon calling ENDLOCAL, calling EXIT, or when execution reaches the end of file (EOF) in your script. The following example shows the difference when local and global variables are set in the script.

Example

```
@echo off
set globalvar=5
SETLOCAL
set var=13145
set /A var=%var% + 5
ENDLOCAL
echo %var%
echo %globalvar%
```

Few key things to note about the above program.

- The 'globalvar' is defined with a global scope and is available throughout the entire script.

- The 'var' variable is defined in a local scope because it is enclosed between a 'SETLOCAL' and 'ENDLOCAL' block. Hence, this variable will be destroyed as soon the 'ENDLOCAL' statement is executed.

Output

The above command produces the following output.

```
13150
5
```

You will notice that the command echo %var% will not yield anything because after the ENDLOCAL statement, the 'var' variable will no longer exist.

Working with Environment Variables

If you have variables that would be used across batch files, then it is always preferable to use environment variables. Once the environment variable is defined, it can be accessed via the % sign. The following example shows how to see the JAVA_HOME defined on a system. The JAVA_HOME variable is a key component that is normally used by a wide variety of applications.

```
@echo off
echo %JAVA_HOME%
```

The output would show the JAVA_HOME directory which would depend from system to system. Following is an example of an output.

```
C:\Atlassian\Bitbucket\4.0.1\jre
```

7. Batch Script – Comments

It’s always a good practice to add comments or documentation for the scripts which are created. This is required for maintenance of the scripts to understand what the script actually does.

For example, consider the following piece of code which has no form of comments. If any average person who has not developed the following script tries to understand the script, it would take a lot of time for that person to understand what the script actually does.

```
ECHO OFF
IF NOT "%OS%"=="Windows_NT" GOTO Syntax
ECHO.%* | FIND "?" >NUL
IF NOT ERRORLEVEL 1 GOTO Syntax
IF NOT [%2]==[] GOTO Syntax
SETLOCAL
SET WSS=
IF NOT [%1]==[] FOR /F "tokens=1 delims=\ " %%A IN ('ECHO.%~1') DO SET WSS=%%A
FOR /F "tokens=1 delims=\ " %%a IN ('NET VIEW ^| FIND /I "\\%WSS%"') DO FOR /F
"tokens=1 delims= " %%A IN ('NBTSTAT -a %%a ^| FIND /I /V "%%a" ^| FIND
"<03>"') DO ECHO.%%a    %%A
ENDLOCAL
GOTO:EOF
ECHO Display logged on users and their workstations.
ECHO Usage:    ACTUSR  [ filter ]
IF     "%OS%"=="Windows_NT" ECHO Where:    filter is the first part of the
computer name^(s^) to be displayed
```

Comments Using the Rem Statement

There are two ways to create comments in Batch Script; one is via the Rem command. Any text which follows the Rem statement will be treated as comments and will not be executed. Following is the general syntax of this statement.

Syntax

```
Rem Remarks
```

where ‘Remarks’ is the comments which needs to be added.

The following example shows a simple way the **Rem** command can be used.

Example

```
@echo off
Rem This program just displays Hello World
set message=Hello World
echo %message%
```

Output

The above command produces the following output. You will notice that the line with the Rem statement will not be executed.

```
Hello World
```

Comments Using the :: Statement

The other way to create comments in Batch Script is via the :: command. Any text which follows the :: statement will be treated as comments and will not be executed. Following is the general syntax of this statement.

Syntax

```
:: Remarks
```

where 'Remarks' is the comment which needs to be added.

The following example shows a simple way the Rem command can be used.

Example

```
@echo off
:: This program just displays Hello World
set message=Hello World
echo %message%
```

Output

The above command produces the following output. You will notice that the line with the :: statement will not be executed.

```
Hello World
```

Note: If you have too many lines of Rem, it could slow down the code, because in the end each line of code in the batch file still needs to be executed.

Let's look at the example of the large script we saw at the beginning of this topic and see how it looks when documentation is added to it.

```
::===============================================================

:: The below example is used to find computer and logged on users

::

::===============================================================

ECHO OFF
:: Windows version check
IF NOT "%OS%"=="Windows_NT" GOTO Syntax
ECHO.%* | FIND "?" >NUL
:: Command line parameter check
IF NOT ERRORLEVEL 1 GOTO Syntax
IF NOT [%2]==[] GOTO Syntax
:: Keep variable local
SETLOCAL
:: Initialize variable
SET WSS=
:: Parse command line parameter
IF NOT [%1]==[] FOR /F "tokens=1 delims=\ " %%A IN ('ECHO.%~1') DO SET WSS=%%A
:: Use NET VIEW and NBTSTAT to find computers and logged on users
FOR /F "tokens=1 delims=\ " %%a IN ('NET VIEW ^| FIND /I "\\%WSS%"') DO FOR /F
"tokens=1 delims= " %%A IN ('NBTSTAT -a %%a ^| FIND /I /V "%%a" ^| FIND
"<03>"') DO ECHO.%%a    %%A
:: Done
ENDLOCAL

GOTO:EOF
:Syntax
ECHO Display logged on users and their workstations.
ECHO Usage:    ACTUSR  [ filter ]
IF     "%OS%"=="Windows_NT" ECHO Where:    filter is the first part of the
computer name^(s^) to be displayed
```

You can now see that the code has become more understandable to users who have not developed the code and hence is more maintainable.

8. Batch Script – Strings

In DOS, a string is an ordered collection of characters, such as "Hello, World!".

Create String

A string can be created in DOS in the following way.

Example

```
@echo off
:: This program just displays Hello World
set message=Hello World
echo %message%
```

Output

The above command produces the following output.

```
Hello World
```

Empty String

An empty string can be created in DOS Scripting by assigning it no value during it's initialization as shown in the following example.

```
Set a=
```

To check for an existence of an empty string, you need to encompass the variable name in square brackets and also compare it against a value in square brackets as shown in the following example.

```
[%a%] == []
```

The following example shows how an empty string can be created and how to check for the existence of an empty string.

Example

```
@echo off

SET a=

SET b=Hello

if [%a%] == [] echo "String A is empty"

if [%b%] == [] echo "String B is empty "
```

Output

The above command produces the following output.

```
String A is empty
```

String Interpolation

String interpolation is a way to construct a new String value from a mix of constants, variables, literals, and expressions by including their values inside a string literal.

In DOS scripting, the string interpolation can be done using the set command and lining up the numeric defined variables or any other literals in one line when using the set command.

The following example shows how a string interpolation can be done with numeric values as well.

Example

```
@echo off
SET a=Hello
SET b=World
SET /A d=50
SET c=%a% and %b% %d%
echo %c%
```

Output

The above command produces the following output.

```
Hello and World 50
```

String Concatenation

You can use the set operator to concatenate two strings or a string and a character, or two characters. Following is a simple example which shows how to use string concatenation.

Example

```
@echo off
SET a=Hello
SET b=World
SET c=%a% and %b%
echo %c%
```

Output

The above command produces the following output.

```
Hello and World
```

String length

In DOS scripting, there is no length function defined for finding the length of a string. There are custom-defined functions which can be used for the same. Following is an example of a custom-defined function for seeing the length of a string.

Example

```
@echo off
set str=Hello World
call :strLen str strlen
echo String is %strlen% characters long
exit /b
:strLen
setlocal enabledelayedexpansion
:strLen_Loop
  if not "!%1:~%len%!"=="" set /A len+=1 & goto :strLen_Loop
(endlocal & set %2=%len%)
goto :eof
```

A few key things to keep in mind about the above program are:

- The actual code which finds the length of string is defined in the :strLen block.
- The length of the string is maintained in the variable len.

Output

The above command produces the following output.

```
11
```

toInt

A variable which has been set as string using the set variable can be converted to an integer using the /A switch which is using the set variable. The following example shows how this can be accomplished.

Example

```
@echo off

set var=13145

set /A var=%var% + 5

echo %var%
```

Output

The above command produces the following output.

```
13150
```

Apart from this, strings have the following implementations which are available. Batch scripts have the following commands which are used to carry out string manipulation in strings.

```
%variable:~num_chars_to_skip%

%variable:~num_chars_to_skip,num_chars_to_keep%
```

This can include negative numbers:

```
%variable:~num_chars_to_skip, -num_chars_to_keep%

%variable:~-num_chars_to_skip,num_chars_to_keep%

%variable:~-num_chars_to_skip,-num_chars_to_keep%
```

Let us discuss the possible string operations that can be performed by using the above commands.

Align Right

This used to align text to the right, which is normally used to improve readability of number columns.

Example

```
@echo off
set x=1000
set y=1
set y=        %y%
echo %x%
set y=%y:~-4%
echo %y%
```

A few key things to note about the above program is:

- Spaces are added to the variable of y, in this case we are adding 9 spaces to the variable of y.
- We are using the ~-4 option to say that we just want to show the last 4 characters of the string y.

Output

The above command produces the following output. The key thing to note is that the value of 2 is aligned to match the units columns when displaying numbers.

```
3000
     2
```

Left String

This is used to extract characters from the beginning of a string.

Example

```
@echo off
set str=Helloworld
echo %str%
set str=%str:~0,5%
echo %str%
```

The key thing to note about the above program is, ~0,5 is used to specify the characters which needs to be displayed. In this case, we are saying character 0 to 5 should be displayed.

Output

The above command produces the following output.

```
Helloworld
Hello
```

Mid String

This is used to extract a substring via the position of the characters in the string.

Example

```
@echo off

set str = Helloworld

echo %str%
```

```
set str=%str:~5,10%

echo %str%
```

The key thing to note about the above program is, ~5,10 is used to specify the characters which needs to be displayed. In this case, we want character 5 to 10 should be displayed.

Output

The above command produces the following output.

```
Helloworld
world
```

Remove

The string substitution feature can also be used to remove a substring from another string.

Example

```
@echo off
set str=Batch scripts is easy. It is really easy.
echo %str%
set str=%str:is =%
echo %str%
```

The key thing to note about the above program is, the 'is' word is being removed from the string using the :'stringtoberemoved' = command.

Output

The above command produces the following output.

```
Batch scripts is easy. It is really easy.
Batch scripts easy. It really easy.
```

Remove Both Ends

This is used to remove the first and the last character of a string.

Example

```
@echo off

set str=Batch scripts is easy. It is really easy
echo %str%
set str=%str:~1,-1%
echo %str%
```

The key thing to note about the above program is, the ~1,-1 is used to remove the first and last character of a string.

Output

The above command produces the following output.

```
Batch scripts is easy. It is really easy

atch scripts is easy. It is really eas
```

Remove All Spaces

This is used to remove all spaces in a string via substitution.

Example

```
@echo off
set str=This string     has      a    lot    of spaces
echo %str%
set str=%str: =%
echo %str%
```

The key thing to note about the above program is, the : = operator is used to remove all spaces from a string.

Output

The above command produces the following output.

```
This string     has      a    lot    of spaces
Thisstringhasalotofspaces
```

Replace a String

To replace a substring with another string use the string substitution feature.

Example

```
@echo off
set str=This message needs changed.
echo %str%
set str=%str:needs=has%
echo %str%
```

The key thing to note about the above program is, the example replaces the word 'needs' with the string 'has' via the statement %str:needs=has%

Output

The above command produces the following output.

```
This message needs changed.
This message has changed.
```

Right String

This is used to extract characters from the end of a string.

Example

```
@echo off
set str=This message needs changed.
echo %str%
set str=%str:~-8%
echo %str%
```

The key thing to note about the above program is, the right hand of the string is extracted by using the ~-'number of characters to extract' operator.

Output

The above command produces the following output.

```
This message needs changed.
changed.
```

9. Batch Script – Arrays

Arrays are not specifically defined as a type in Batch Script but can be implemented. The following things need to be noted when arrays are implemented in Batch Script.

- Each element of the array needs to be defined with the set command.
- The 'for' loop would be required to iterate through the values of the array.

Creating an Array

An array is created by using the following set command.

```
set a[0]=1
```

Where 0 is the index of the array and 1 is the value assigned to the first element of the array.

Another way to implement arrays is to define a list of values and iterate through the list of values. The following example show how this can be implemented.

Example

```
@echo off
set list=1 2 3 4
(for %%a in (%list%) do (
   echo %%a
))
```

Output

The above command produces the following output.

```
1
2
3
4
```

Accessing Arrays

You can retrieve a value from the array by using subscript syntax, passing the index of the value you want to retrieve within square brackets immediately after the name of the array.

Example

```
@echo off
set a[0]=1
echo %a[0]%
```

In this example, the index starts from 0 which means the first element can be accessed using index as 0, the second element can be accessed using index as 1 and so on. Let's check the following example to create, initialize and access arrays:

```
@echo off
set a[0]=1
set a[1]=2
set a[2]=3
echo The first element of the array is %a[0]%
echo The second element of the array is %a[1]%
echo The third element of the array is %a[2]%
```

The above command produces the following output.

```
The first element of the array is 1
The second element of the array is 2
The third element of the array is 3
```

Modifying an Array

To add an element to the end of the array, you can use the set element along with the last index of the array element.

Example

```
@echo off
set a[0]=1
set a[1]=2
set a[2]=3
Rem Adding an element at the end of an array
Set a[3]=4
echo The last element of the array is %a[3]%
```

The above command produces the following output.

```
The last element of the array is 4
```

You can modify an existing element of an Array by assigning a new value at a given index as shown in the following example:

```
@echo off
set a[0]=1
set a[1]=2
set a[2]=3
Rem Setting the new value for the second element of the array
Set a[1]=5
echo The new value of the second element of the array is %a[1]%
```

The above command produces the following output.

```
The new value of the second element of the array is 5
```

Iterating Over an Array

Iterating over an array is achieved by using the 'for' loop and going through each element of the array. The following example shows a simple way that an array can be implemented.

```
@echo off
setlocal enabledelayedexpansion
set topic[0]=comments
set topic[1]=variables
set topic[2]=Arrays
set topic[3]=Decision making
set topic[4]=Time and date
set topic[5]=Operators
for /l %%n in (0,1,5) do (
echo !topic[%%n]!
)
```

Following things need to be noted about the above program:

- Each element of the array needs to be specifically defined using the set command.
- The 'for' loop with the /L parameter for moving through ranges is used to iterate through the array.

Output

The above command produces the following output.

```
Comments
variables
Arrays
Decision making
Time and date
Operators
```

Length of an Array

The length of an array is done by iterating over the list of values in the array since there is no direct function to determine the number of elements in an array.

```
@echo off
set Arr[0]=1
set Arr[1]=2
set Arr[2]=3
set Arr[3]=4
set "x=0"
:SymLoop
if defined Arr[%x%] (
    call echo %%Arr[%x%]%%
    set /a "x+=1"
    GOTO :SymLoop
)
echo "The length of the array is" %x%
```

Output

The above command produces the following output.

```
The length of the array is 4
```

Creating Structures in Arrays

Structures can also be implemented in batch files using a little bit of an extra coding for implementation. The following example shows how this can be achieved.

Example

```
@echo off
set len=3
set obj[0].Name=Joe
set obj[0].ID=1
set obj[1].Name=Mark
set obj[1].ID=2
set obj[2].Name=Mohan
set obj[2].ID=3
set i=0
:loop
if %i% equ %len% goto :eof
set cur.Name=
set cur.ID=
for /f "usebackq delims==. tokens=1-3" %%j in (`set obj[%i%]`) do (
    set cur.%%k=%%l
)
echo Name=%cur.Name%
echo Value=%cur.ID%
set /a i=%i%+1
goto loop
```

The following key things need to be noted about the above code.

- Each variable defined using the set command has 2 values associated with each index of the array.

- The variable **i** is set to 0 so that we can loop through the structure will the length of the array which is 3.

- We always check for the condition on whether the value of **i** is equal to the value of **len** and if not, we loop through the code.

- We are able to access each element of the structure using the obj[%i%] notation.

Output

The above command produces the following output.

```
Name=Joe
Value=1
Name=Mark
Value=2
Name=Mohan
Value=3
```

10. Batch Script – Decision Making

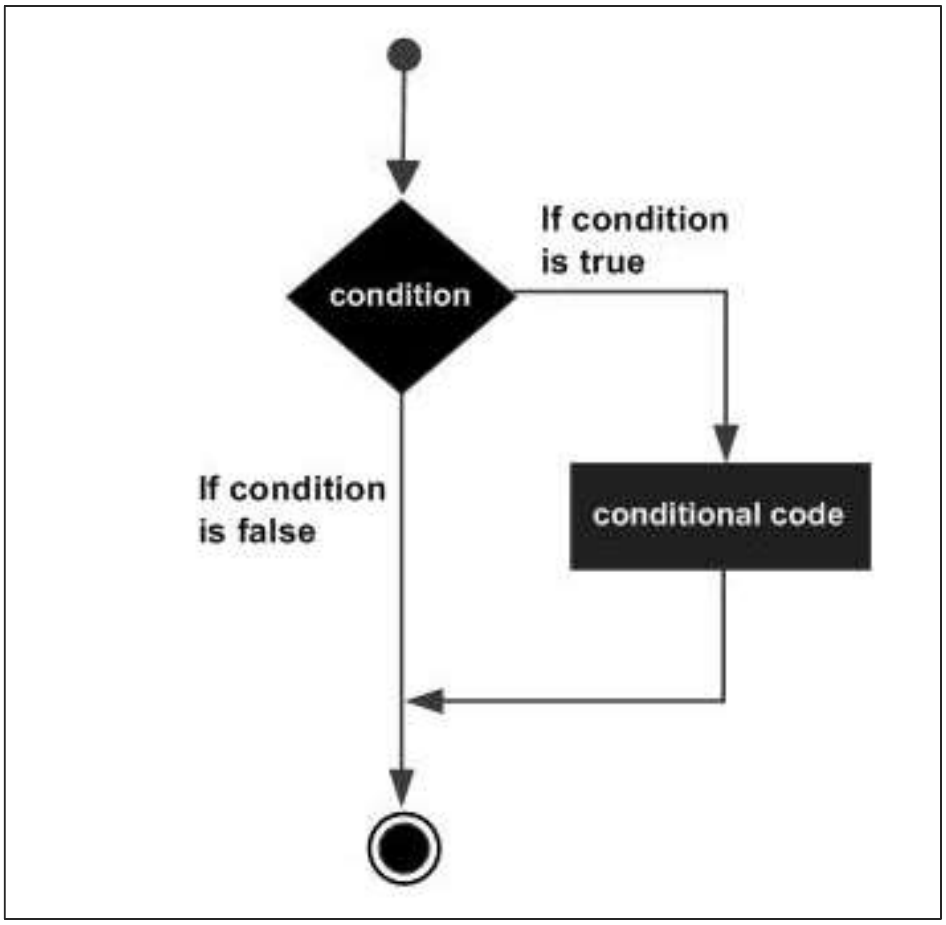
Decision-making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be **true**, and optionally, other statements to be executed if the condition is determined to be **false**.

If Statement

The first decision-making statement is the 'if' statement. The general form of this statement in Batch Script is as follows:

```
if(condition) do_something
```

The general working of this statement is that first a condition is evaluated in the 'if' statement. If the condition is true, it then executes the statements. The following diagram shows the flow of the **if** statement.



Checking Variables

One of the common uses for the 'if' statement in Batch Script is for checking variables which are set in batch script itself. The evaluation of the 'if' statement can be done for both strings and numbers.

Checking Integer Variables

The following example shows how the 'if' statement can be used for numbers.

Example

```
@echo off
SET /A a=5
SET /A b=10
SET /A c=%a% + %b%
if %c% == 15 echo "The value of variable c is 15"
if %c% == 10 echo "The value of variable c is 10"
```

The key thing to note about the above program is:

- The first 'if' statement checks if the value of the variable c is 15. If so, then it echo's a string to the command prompt.

- Since the condition in the statement - if %c% == 10 echo "The value of variable **c** is 10 evaluates to false, the echo part of the statement will not be executed.

Output

The above command produces the following output.

```
15
```

Checking String Variables

The following example shows how the 'if' statement can be used for strings.

Example

```
@echo off
SET str1=String1
SET str2=String2
if %str1% == String1 echo "The value of variable String1"
if %str2% == String3 echo "The value of variable c is String3"
```

The key thing to note about the above program is:

- The first 'if' statement checks if the value of the variable str1 contains the string "String1". If so, then it echo's a string to the command prompt.

- Since the condition of the second 'if' statement evaluates to false, the echo part of the statement will not be executed.

Output

The above command produces the following output.

```
"The value of variable String1"
```

Note: One key thing to note is that the evaluation in the 'if' statement is "case-sensitive". The same program as above is modified a little as shown in the following example. In the first statement, we have changed the comparison criteria. Because of the different casing, the output of the following program would yield nothing.

```
@echo off
SET str1=String1
SET str2=String2
if %str1% == StrinG1 echo "The value of variable String1"
if %str2% == String3 echo "The value of variable c is String3"
```

Checking Command Line Arguments

Another common use of the 'if' statement is used to check for the values of the command line arguments which are passed to the batch files. The following example shows how the 'if' statement can be used to check for the values of the command line arguments.

```
@echo off
echo %1
echo %2
echo %3
if %1% == 1 echo "The value is 1"
if %2% == 2 echo "The value is 2"
if %3% == 3 echo "The value is 3"
```

The key thing to note about the above program is:

- The above program assumes that 3 command line arguments will be passed when the batch script is executed.

- A comparison is done for each command line argument against a value. If the criteria passes then a string is sent as the output.

Output

If the above code is saved in a file called test.bat and the program is executed as

```
test.bat 1 2 3
```

Following will be the output of the above program.

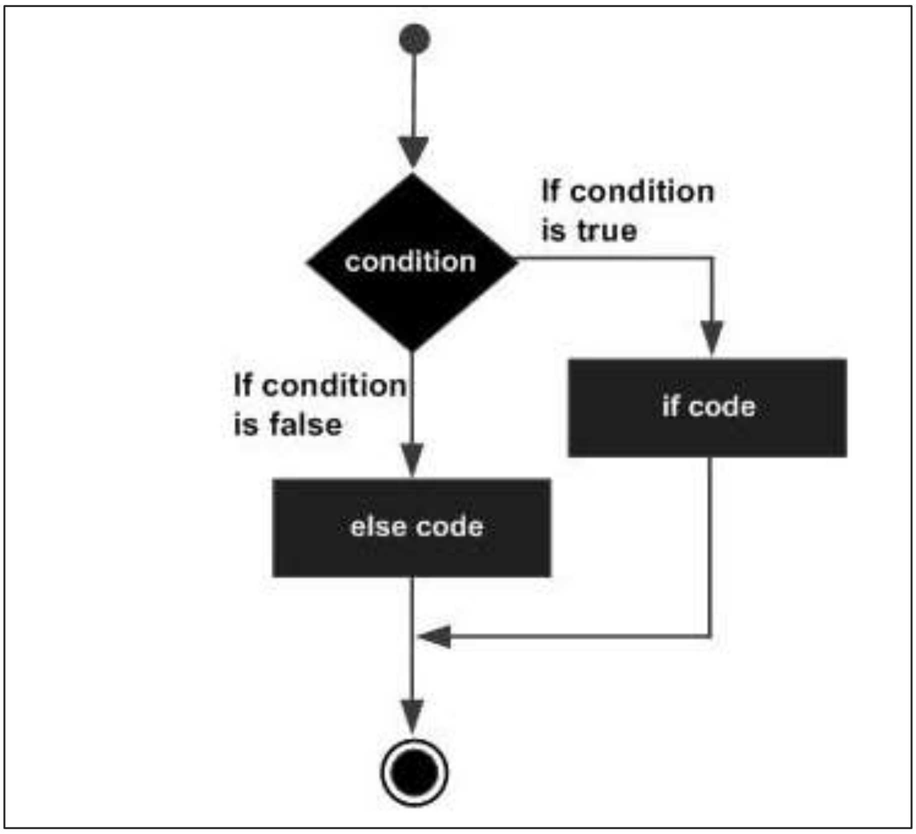
```
1
2
3
"The value is 1"
"The value is 2"
"The value is 3"
```

If/else Statement

The next decision making statement is the If/else statement. Following is the general form of this statement.

```
If (condition) (do_something) ELSE (do_something_else)
```

The general working of this statement is that first a condition is evaluated in the 'if' statement. If the condition is true, it then executes the statements thereafter and stops before the else condition and exits out of the loop. If the condition is false, it then executes the statements in the else statement block and then exits the loop. The following diagram shows the flow of the 'if' statement.



Checking Variables

Just like the 'if' statement in Batch Script, the if-else can also be used for checking variables which are set in Batch Script itself. The evaluation of the 'if' statement can be done for both strings and numbers.

Checking Integer Variables

The following example shows how the 'if' statement can be used for numbers.

Example

```
@echo off
SET /A a=5
SET /A b=10
SET /A c=%a% + %b%
if %c% == 15 (echo "The value of variable c is 15") else (echo "Unknown value")
if %c% == 10 (echo "The value of variable c is 10") else (echo "Unknown value")
```

The key thing to note about the above program is :

- Each 'if else' code is placed in the brackets (). If the brackets are not placed to separate the code for the 'if and else' code, then the statements would not be valid proper if else statements.

- In the first 'if else' statement, the if condition would evaluate to true.

- In the second 'if else' statement, the else condition will be executed since the criteria would be evaluated to false.

Output

The above command produces the following output.

```
"The value of variable c is 15"
"Unknown value"
```

Checking String Variables

The same example can be repeated for strings. The following example shows how the 'if else' statement can be used to strings.

Example

```
@echo off
SET str1=String1
SET str2=String2
if %str1% == String1 (echo "The value of variable String1") else (echo "Unknown
value")
if %str2% == String3 (echo "The value of variable c is String3") else (echo
"Unknown value")
```

The key thing to note about the above program is:

- The first 'if' statement checks if the value of the variable str1 contains the string "String1". If so, then it echo's a string to the command prompt.

- Since the condition of the second 'if' statement evaluates to false, the echo part of the statement will not be executed.

Output

The above command produces the following output.

```
"The value of variable String1"
"Unknown value"
```

Checking Command Line Arguments

The ‘if else’ statement can also be used for checking of command line arguments. The following example show how the ‘if’ statement can be used to check for the values of the command line arguments.

```
@echo off
echo %1
echo %2
echo %3
if %1% == 1 (echo "The value is 1") else (echo "Unknown value")
if %2% == 2 (echo "The value is 2") else (echo "Unknown value")
if %3% == 3 (echo "The value is 3") else (echo "Unknown value")
```

Output

If the above code is saved in a file called test.bat and the program is executed as

```
test.bat 1 2 4
```

Following will be the output of the above program.

```
1
2
4
"The value is 1"
"The value is 2"
"Unknown value"
```

if defined

A special case for the ‘if’ statement is the "if defined", which is used to test for the existence of a variable. Following is the general syntax of the statement.

```
if defined somevariable somecommand
```

Following is an example of how the ‘if defined’ statement can be used.

Example

```
@echo off
SET str1=String1
SET str2=String2
if defined str1 echo "Variable str1 is defined"

if defined str3 (echo "Variable str3 is defined") else (echo "Variable str3 is
not defined")
```

Output

The above command produces the following output.

```
"Variable str1 is defined"
"Variable str3 is not defined"
```

if exists

Another special case for the ‘if’ statement is the "if exists ", which is used to test for the existence of a file. Following is the general syntax of the statement.

```
If exist somefile.ext do_something
```

Following is an example of how the ‘if exists’ statement can be used.

Example

```
@echo off
if exist C:\set2.txt echo "File exists"
if exist C:\set3.txt (echo "File exists") else (echo "File does not exist")
```

Output

Let’s assume that there is a file called set2.txt in the C drive and that there is no file called set3.txt. Then, following will be the output of the above code.

```
"File exists"

"File does not exist"
```

Nested If Statements

Sometimes, there is a requirement to have multiple ‘if’ statement embedded inside each other. Following is the general form of this statement.

```
if(condition1) if (condition2) do_something
```

So only if condition1 and condition2 are met, will the code in the do_something block be executed.

Following is an example of how the nested if statements can be used.

Example

```
@echo off

SET /A a=5

SET /A b=10

if %a% == 5 if %b% == 10 echo "The value of the variables are correct"
```

Output

The above command produces the following output.

```
"The value of the variables are correct"
```

If errorlevel

Yet another special case is "if errorlevel", which is used to test the exit codes of the last command that was run. Various commands issue integer exit codes to denote the status of the command. Generally, commands pass 0 if the command was completed successfully and 1 if the command failed.

Following is the general syntax of this statement.

```
if errorlevel n somecommand
```

where "n" is one of the integer exit codes.

Goto Statement

Generally, the execution of a batch file proceeds line-by-line with the command(s) on each line being run in turn. However, it is often desirable to execute a particular section of a batch file while skipping over other parts. The capability to hop to a particular section is provided by the appropriately named "goto" command (written as one word). The target section is labeled with a line at the beginning that has a name with a leading colon. Thus the script looks like:

```
...
goto :label
...some commands
:label
...some other commands
```

Execution will skip over "some commands" and start with "some other commands". The label can be a line anywhere in the script, including before the "goto" command. "Goto" commands often occur in "if" statements. For example, you might have a command of the type:

```
if (condition) goto :label
```

Following is an example of how the goto statement can be used.

Example

```
@echo off
SET /A a=5
SET /A b=10
if %a% == 5 goto :labela
if %b% == 10 goto :labelb
:labela
echo "The value of a is 5"
:labelb
echo "The value of a is 10"
```

The key thing to note about the above program is:

- The code statements for the label should be on the next line after the declaration of the label.
- You can define multiple goto statements and their corresponding labels in a batch file.
- The label declarations can be anywhere in the file.

Output

The above command produces the following output.

```
"The value of a is 5"

"The value of a is 10"
```

11. Batch Script – Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations.

In batch script, the following types of operators are possible.

- Arithmetic operators
- Relational operators
- Logical operators
- Assignment operators
- Bitwise operators

Arithmetic Operators

Batch Script language supports the normal Arithmetic operators as any language. Following are the Arithmetic operators available.

Operator	Description	Example
+	Addition of two operands	1 + 2 will give 3
–	Subtracts second operand from the first	2 – 1 will give 1
*****	Multiplication of both operands	2 * 2 will give 4
/	Division of the numerator by the denominator	3 / 2 will give 1.5
%	Modulus operator and remainder of after an integer/float division	3 % 2 will give 1

The following code snippet shows how the various operators can be used.

```
@echo off
SET /A a=5
SET /A b=10
SET /A c=%a%+%b%
echo %c%
SET /A c=%a%-%b%
echo %c%
SET /A c=%b%*%a%
echo %c%
```

```
SET /A c=%b%/%a%
echo %c%
SET /A c=%b% %% %a%
echo %c%
```

Output

The above command produces the following output.

```
15
-5
50
2
0
```

Relational Operators

Relational operators allow of the comparison of objects. Below are the relational operators available.

Operator	Description	Example
EQU	Tests the equality between two objects	2 EQU 2 will give true
NEQ	Tests the difference between two objects	3 NEQ 2 will give true
LSS	Checks to see if the left object is less than the right operand	2 LSS 3 will give true
LEQ	Checks to see if the left object is less than or equal to the right operand	2 LEQ 3 will give true
GTR	Checks to see if the left object is greater than the right operand	3 GTR 2 will give true
GEQ	Checks to see if the left object is greater than or equal to the right operand	3 GEQ 2 will give true

The following code snippet shows how the various operators can be used.

```
@echo off
SET /A a=5
SET /A b=10
if %a% EQU %b% echo A is equal to than B
if %a% NEQ %b% echo A is not equal to than B
if %a% LSS %b% echo A is less than B
if %a% LEQ %b% echo A is less than or equal B
if %a% GTR %b% echo A is greater than B
if %a% GEQ %b% echo A is greater than or equal to B
```

Output

The above command produces the following output.

```
A is not equal to than B

A is less than B

A is less than or equal B
```

Logical Operators

Logical operators are used to evaluate Boolean expressions. Following are the logical operators available.

The batch language is equipped with a full set of Boolean logic operators like AND, OR, XOR, but only for binary numbers. Neither are there any values for TRUE or FALSE. The only logical operator available for conditions is the NOT operator.

Operator	Description
AND	This is the logical "and" operator
OR	This is the logical "or" operator
NOT	This is the logical "not" operator

The easiest way to implement the AND/OR operator for non-binary numbers is to use the nested IF condition. The following example shows how this can be implemented.

```
@echo off

SET /A a=5

SET /A b=10

IF %a% LSS 10 (IF %b% GTR 0 (ECHO %a% is less than 10 AND %b% is greater than
0))
```

Output

The above command produces the following output.

```
5 is less than 10 AND 10 is greater than 0
```

Following is an example of the AND operation that can be implemented using the IF statement.

```
@echo off
SET /A a=5
SET /A b=10
IF %a% GEQ 10 (
     IF %b% LEQ 0 (
            ECHO %a% is NOT less than 10 OR %b% is NOT greater than 0
     ) ELSE (
            ECHO %a% is less than 10 OR %b% is greater than 0
     )
) ELSE (
     ECHO %a% is less than 10 OR %b% is greater than 0
)
```

Output

The above command produces the following output.

```
5 is less than 10 AND 10 is greater than 0
```

Following is an example of how the NOT operator can be used.

```
@echo off
SET /A a=5
IF NOT %a%==6 echo "A is not equal to 6"
```

Output

The above command produces the following output.

```
"A is equal to 5"
```

Assignment Operators

Batch Script language also provides assignment operators. Following are the assignment operators available.

Operator	Description	Example
+=	This adds right operand to the left operand and assigns the result to left operand	Set /A a = 5 a+=3

		Output will be 8
-=	This subtracts the right operand from the left operand and assigns the result to the left operand	Set /A a = 5 a-=3 Output will be 2
***=**	This multiplies the right operand with the left operand and assigns the result to the left operand	Set /A a = 5 a*=3 Output will be 15
/=	This divides the left operand with the right operand and assigns the result to the left operand	Set /A a = 6 a/=3 Output will be 2
%=	This takes modulus using two operands and assigns the result to the left operand	Set /A a = 5 a%=3 Output will be 2

The following code snippet shows how the various operators can be used.

```
@echo off
SET /A a=5
SET /A a +=5
echo %a%
SET /A a -=5
echo %a%
SET /A a *=5
echo %a%
SET /A a /=5
echo %a%
SET /A a %=5
echo %a%
```

Output

The above command produces the following output.

```
10
5
25
5
5
```

Bitwise Operators

Bitwise operators are also possible in Batch Script. Following are the operators available.

Operator	Description
&	This is the bitwise "and" operator
|	This is the bitwise "or" operator
^	This is the bitwise "xor" or Exclusive or operator

Following is the truth table showcasing these operators.

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

The following code snippet shows how the various operators can be used.

```
@echo off
SET /A "Result = 48 & 23"
echo %Result%
SET /A "Result = 16 | 16"
echo %Result%
SET /A "Result = 31 ^ 15"
echo %Result%
```

Output

The above command produces the following output.

```
16

16

16
```

Redirection

Redirection is a concept of taking the output of a command and re-directing that output to a different output media. The following commands are available for re-direction.

- **command > filename** – Redirect command output to a file.
- **command >> filename** – APPEND into a file.
- **command < filename** – Type a text file and pass the text to command.
- **command 2> file** – Write standard error of command to file (OS/2 and NT).
- **command 2>> file** – Append standard error of command to file (OS/2 and NT).
- **commandA | commandB** – Redirect standard output of commandA to standard input of command.

The following code snippet shows how the various redirection operations can be used.

command > filename

This command redirects command output to a file.

Example

```
@echo off
ipconfig>C:\details.txt
```

Output

The output of the above program would be that all the details of the ipconfig command will be sent to the file C:\details.txt. If you open the above file, you might see the information similar to the one as the following.

```
Windows IP Configuration

Wireless LAN adapter Local Area Connection* 11:

   Media State . . . . . . . . . . . : Media disconnected

   Connection-specific DNS Suffix  . :

Ethernet adapter Ethernet:

   Media State . . . . . . . . . . . : Media disconnected

   Connection-specific DNS Suffix  . :

Wireless LAN adapter Wi-Fi:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
Tunnel adapter Teredo Tunneling Pseudo-Interface:
   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
```

command >> filename

This command appends the output of the command into a file.

Example

```
@echo off
systeminfo>>C:\details.txt
```

Output

The output of the above program would be that all the details of the systeminfo command will be appended to the file C:\details.txt. if you open the above file you might see the information similar to the one as the following.

```
Windows IP Configuration
Wireless LAN adapter Local Area Connection* 11:
  Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
Ethernet adapter Ethernet:
   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
Wireless LAN adapter Wi-Fi:
   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
Tunnel adapter Teredo Tunneling Pseudo-Interface:
   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
Host Name:                 WIN-50GP30FGO75
OS Name:                   Microsoft Windows Server 2012 R2 Standard
OS Version:                6.3.9600 N/A Build 9600
OS Manufacturer:           Microsoft Corporation
OS Configuration:          Standalone Server
OS Build Type:             Multiprocessor Free
Registered Owner:          Windows User
Registered Organization:
```

```
Product ID:                 00252-70000-00000-AA535
Original Install Date:      12/13/2015, 12:10:16 AM
System Boot Time:           12/30/2015, 5:52:11 AM
System Manufacturer:        LENOVO
System Model:               20287
System Type:                x64-based PC
```

command < filename

This command types a text file and passes the text to command.

Example

```
@echo off
SORT < Example.txt
```

Output

If you define a file called Example.txt which has the following data.

```
4
3
2
1
```

The output of the above program would be

```
1
2
3
4
```

command 2> file

This command writes the standard error of command to file (OS/2 and NT).

Example

```
DIR C:\ >List_of_C.txt 2>errorlog.txt
```

In the above example, if there is any error in processing the command of the directory listing of C, then it will be sent to the log file errorlog.txt.

command 2>> file

Appends the standard error of command to file (OS/2 and NT).

Example

```
DIR C:\ >List_of_C.txt 2>errorlog.txt
DIR D:\ >List_of_C.txt 2>>errorlog.txt
```

In the above example, if there is any error in processing the command of the directory listing of D, then it will be appended to the log file errorlog.txt.

commandA | commandB

This command redirects standard output of commandA to standard input of command.

Example

```
Echo y | del *.txt
```

Output

The above command will pass the option of ‘y’ which is the value of ‘Yes’ to the command of del. This will cause the deletion of all files with the extension of txt.

12. Batch Script – DATE and TIME

The date and time in DOS Scripting have the following two basic commands for retrieving the date and time of the system.

DATE

This command gets the system date.

Syntax

```
DATE
```

Example

```
@echo off
echo %DATE%
```

Output

The current date will be displayed in the command prompt. For example,

```
Mon 12/28/2015
```

TIME

This command sets or displays the time.

Syntax

```
TIME
```

Example

```
@echo off
echo %TIME%
```

Output

The current system time will be displayed. For example,

```
22:06:52.87
```

Following are some implementations which can be used to get the date and time in different formats.

Date in Format Year-Month-Day

Example

```
@echo off
echo/Today is: %year%-%month%-%day%
goto :EOF
setlocal ENABLEEXTENSIONS
set t=2&if "%date%z" LSS "A" set t=1
for /f "skip=1 tokens=2-4 delims=(-)" %%a in ('echo/^|date') do (
  for /f "tokens=%t%-4 delims=.-/ " %%d in ('date/t') do (
    set %%a=%%d&set %%b=%%e&set %%c=%%f))
endlocal&set %1=%yy%&set %2=%mm%&set %3=%dd%&goto :EOF
```

Output

The above command produces the following output.

```
Today is: 2015-12-30
```

13. Batch Script – Input / Output

There are three universal “files” for keyboard input, printing text on the screen and printing errors on the screen. The “Standard In” file, known as **stdin**, contains the input to the program/script. The “Standard Out” file, known as **stdout**, is used to write output for display on the screen. Finally, the “Standard Err” file, known as **stderr**, contains any error messages for display on the screen.

Each of these three standard files, otherwise known as the standard streams, are referenced using the numbers 0, 1, and 2. Stdin is file 0, stdout is file 1, and stderr is file 2.

Redirecting Output (Stdout and Stderr)

One common practice in batch files is sending the output of a program to a log file. The > operator sends, or redirects, stdout or stderr to another file. The following example shows how this can be done.

```
Dir C:\ > list.txt
```

In the above example, the **stdout** of the command Dir C:\ is redirected to the file list.txt.

If you append the number 2 to the redirection filter, then it would redirect the **stderr** to the file lists.txt.

```
Dir C:\ 2> list.txt
```

One can even combine the **stdout** and **stderr** streams using the file number and the ‘&’ prefix. Following is an example.

```
DIR C:\ > lists.txt 2>&1
```

Suppressing Program Output

The pseudo file NUL is used to discard any output from a program. The following example shows that the output of the command DIR is discarded by sending the output to NUL.

```
Dir C:\ > NUL
```

Stdin

To work with the Stdin, you have to use a workaround to achieve this. This can be done by redirecting the command prompt’s own stdin, called CON.

The following example shows how you can redirect the output to a file called lists.txt. After you execute the below command, the command prompt will take all the input entered by user till it gets an EOF character. Later, it sends all the input to the file lists.txt.

```
TYPE CON > lists.txt
```

14. Batch Script – Return Code

By default when a command line execution is completed it should either return zero when execution succeeds or non-zero when execution fails. When a batch script returns a non-zero value after the execution fails, the non-zero value will indicate what is the error number. We will then use the error number to determine what the error is about and resolve it accordingly.

Following are the common exit code and their description.

Error Code	Description
0	Program successfully completed.
1	Incorrect function. Indicates that Action has attempted to execute non-recognized command in Windows command prompt cmd.exe.
2	The system cannot find the file specified. Indicates that the file cannot be found in specified location.
3	The system cannot find the path specified. Indicates that the specified path cannot be found.
5	Access is denied. Indicates that user has no access right to specified resource.
9009 0x2331	Program is not recognized as an internal or external command, operable program or batch file. Indicates that command, application name or path has been misspelled when configuring the Action.
221225495 0xC0000017 -1073741801	Not enough virtual memory is available. It indicates that Windows has run out of memory.
3221225786 0xC000013A -1073741510	The application terminated as a result of a CTRL+C. Indicates that the application has been terminated either by the user's keyboard input CTRL+C or CTRL+Break or closing command prompt window.
3221225794 0xC0000142 -1073741502	The application failed to initialize properly. Indicates that the application has been launched on a Desktop to which the current user has no access rights. Another possible cause is that either gdi32.dll or user32.dll has failed to initialize.

Error Level

The environmental variable %ERRORLEVEL% contains the return code of the last executed program or script.

By default, the way to check for the ERRORLEVEL is via the following code.

Syntax

```
IF %ERRORLEVEL% NEQ 0 (
DO_Something
)
```

It is common to use the command EXIT /B %ERRORLEVEL% at the end of the batch file to return the error codes from the batch file.

EXIT /B at the end of the batch file will stop execution of a batch file.

Use EXIT /B < exitcodes > at the end of the batch file to return custom return codes.

Environment variable %ERRORLEVEL% contains the latest errorlevel in the batch file, which is the latest error codes from the last command executed. In the batch file, it is always a good practice to use environment variables instead of constant values, since the same variable get expanded to different values on different computers.

Let’s look at a quick example on how to check for error codes from a batch file.

Example

Let’s assume we have a batch file called Find.cmd which has the following code. In the code, we have clearly mentioned that we if don’t find the file called lists.txt then we should set the errorlevel to 7. Similarly, if we see that the variable userprofile is not defined then we should set the errorlevel code to 9.

```
if not exist c:\lists.txt exit 7
if not defined userprofile exit 9
exit 0
```

Let’s assume we have another file called App.cmd that calls Find.cmd first. Now, if the Find.cmd returns an error wherein it sets the errorlevel to greater than 0 then it would exit the program. In the following batch file, after calling the Find.cnd find, it actually checks to see if the errorlevel is greater than 0.

```
Call Find.cmd

if errorlevel gtr 0 exit
echo “Successful completion”
```

Output

In the above program, we can have the following scenarios as the output:

- If the file c:\lists.txt does not exist, then nothing will be displayed in the console output.

- If the variable userprofile does not exist, then nothing will be displayed in the console output.

- If both of the above condition passes then the string “Successful completion” will be displayed in the command prompt.

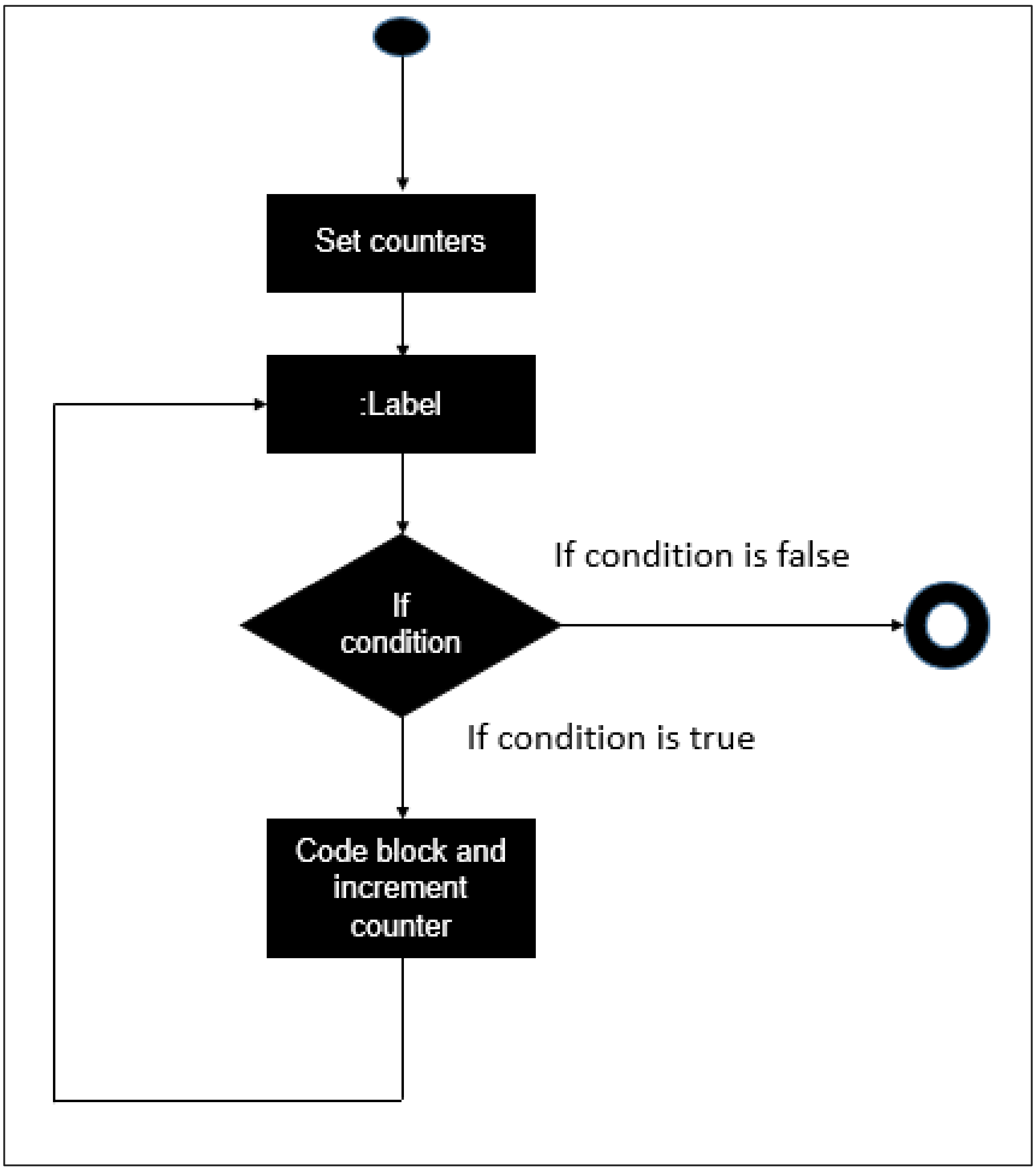
Loops

In the decision making chapter, we have seen statements which have been executed one after the other in a sequential manner. Additionally, implementations can also be done in Batch Script to alter the flow of control in a program's logic. They are then classified into flow of control statements.

While Statement Implementation

There is no direct while statement available in Batch Script but we can do an implementation of this loop very easily by using the if statement and labels.

The following diagram shows the diagrammatic explanation of this loop.



The first part of the while implementation is to set the counters which will be used to control the evaluation of the 'if' condition. We then define our label which will be used to embody the entire code for the while loop implementation. The 'if' condition evaluates an expression. If the expression evaluates to true, the code block is executed. If the condition evaluates to false then the loop is exited. When the code block is executed, it will return back to the label statement for execution again.

Following is the syntax of the general implementation of the while statement.

```
Set counters
:label
If (expression)
(
Do_something
Increment counter
Go back to :label
)
```

- The entire code for the while implementation is placed inside of a label.
- The counter variables must be set or initialized before the while loop implementation starts.
- The expression for the while condition is done using the ‘if’ statement. If the expression evaluates to true then the relevant code inside the ‘if’ loop is executed.
- A counter needs to be properly incremented inside of ‘if’ statement so that the while implementation can terminate at some point in time.
- Finally, we will go back to our label so that we can evaluate our ‘if’ statement again.

Following is an example of a while loop statement.

Example

```
@echo off
SET /A "index=1"
SET /A "count=5"
:while
if %index% leq %count% (
    echo The value of index is %index%
    SET /A "index = index + 1"
    goto :while
)
```

In the above example, we are first initializing the value of an index integer variable to 1. Then our condition in the ‘if’ loop is that we are evaluating the condition of the expression to be that index should it be less than the value of the count variable. Till the value of index is less than 5, we will print the value of index and then increment the value of index.

Output

The above command produces the following output.

```
The value of index is 1
The value of index is 2
The value of index is 3
The value of index is 4
The value of index is 5
```

For Statement - List Implementations

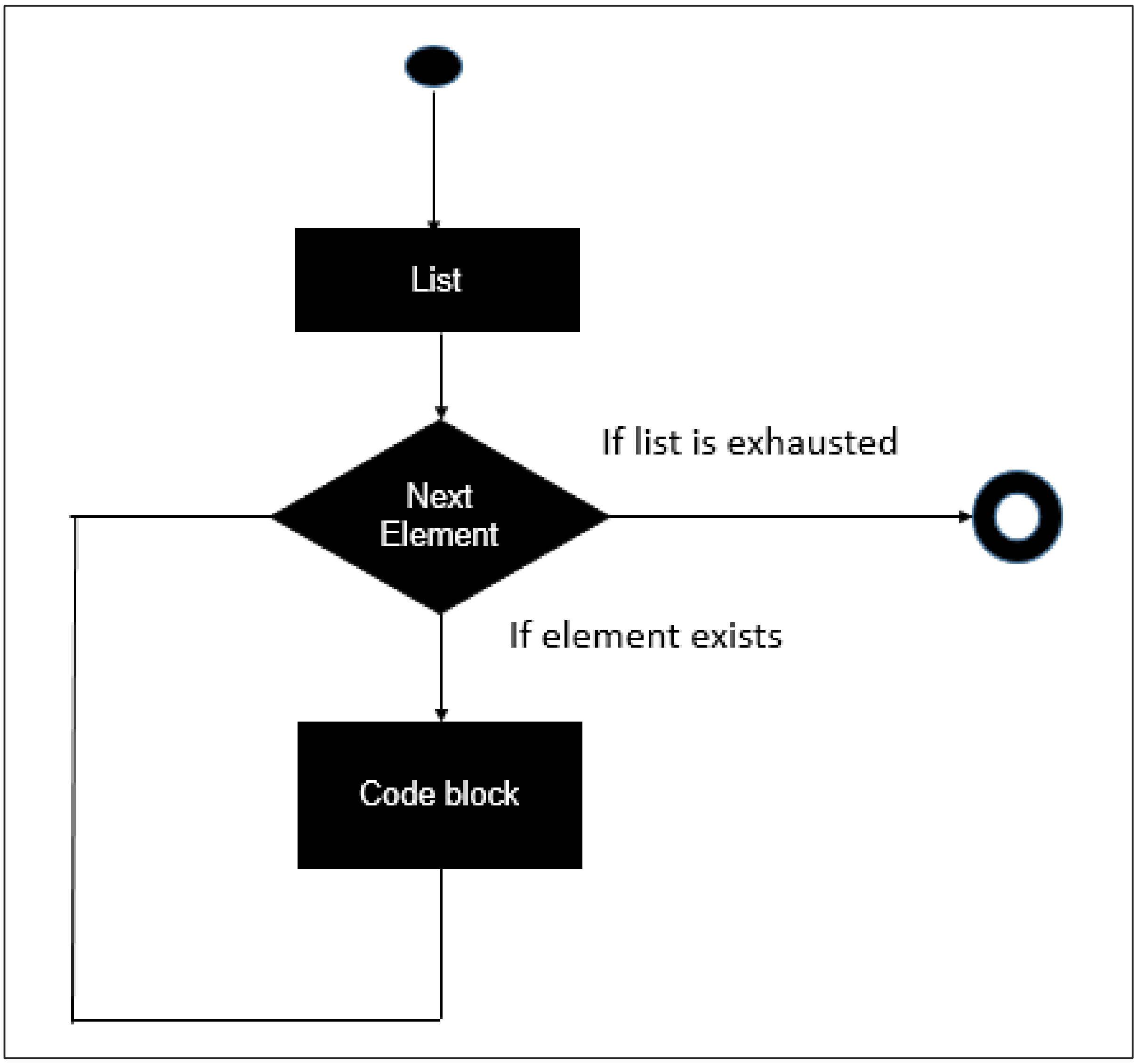
The "FOR" construct offers looping capabilities for batch files. Following is the common construct of the 'for' statement for working with a list of values.

```
FOR %%variable IN list DO do_something
```

The classic 'for' statement consists of the following parts:

- Variable declaration – This step is executed only once for the entire loop and used to declare any variables which will be used within the loop. In Batch Script, the variable declaration is done with the %% at the beginning of the variable name.
- List – This will be the list of values for which the 'for' statement should be executed.
- The do_something code block is what needs to be executed for each iteration for the list of values.

The following diagram shows the diagrammatic explanation of this loop.



Following is an example of how the 'goto' statement can be used.

Example

```
@echo off
FOR %%F IN (1 2 3 4 5) DO echo %%F
```

The key thing to note about the above program is:

- The variable declaration is done with the %% sign at the beginning of the variable name.
- The list of values is defined after the IN clause.
- The do_something code is defined after the echo command. Thus for each value in the list, the echo command will be executed.

Output

The above program produces the following output.

```
1
2
3
4
5
```

Looping through Ranges

The 'for' statement also has the ability to move through a range of values. Following is the general form of the statement.

```
FOR /L %%variable IN (lowerlimit,Increment,Upperlimit) DO do_something
```

Where

- The /L switch is used to denote that the loop is used for iterating through ranges.
- Variable declaration – This step is executed only once for the entire loop and used to declare any variables which will be used within the loop. In Batch Script, the variable declaration is done with the %% at the beginning of the variable name.
- The IN list contains of 3 values. The lowerlimit, the increment, and the upperlimit. So, the loop would start with the lowerlimit and move to the upperlimit value, iterating each time by the Increment value.
- The do_something code block is what needs to be executed for each iteration.

Following is an example of how the looping through ranges can be carried out.

Example

```
@ECHO OFF
FOR /L %%X IN (0,1,5) DO ECHO %%X
```

Output

The above program produces the following output.

```
0
1
2
3
4
5
```

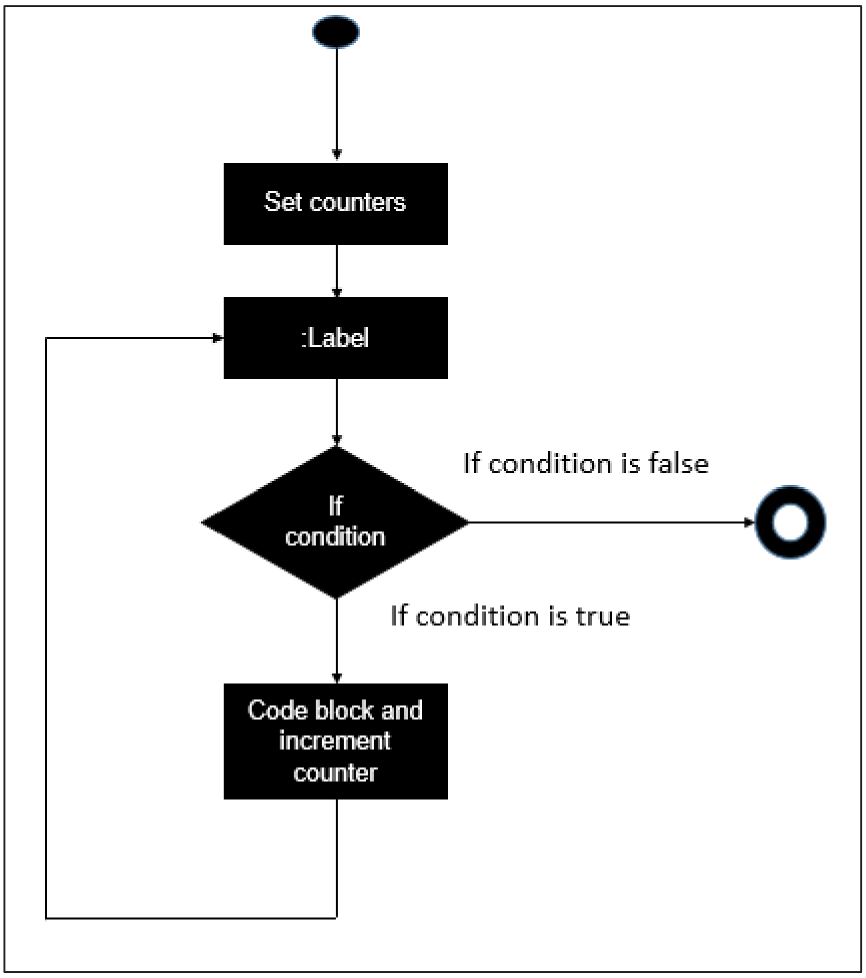
Classic for Loop Implementation

Following is the classic 'for' statement which is available in most programming languages.

```
for(variable declaration;expression;Increment)
{
statement #1
statement #2
…
}
```

The Batch Script language does not have a direct 'for' statement which is similar to the above syntax, but one can still do an implementation of the classic 'for' loop statement using if statements and labels.

Following is the general flow of the classic 'for' loop statement.



Let's look at the general syntax implementation of the classic for loop in Batch Script.

```
Set counter
:label
If (expression) exit loop
```

```
Do_something

Increment counter

Go back to :label
```

- The entire code for the 'for' implementation is placed inside of a label.
- The counters variables must be set or initialized before the 'for' loop implementation starts.
- The expression for the 'for' loop is done using the 'if' statement. If the expression evaluates to be true then an exit is executed to come out of the loop.
- A counter needs to be properly incremented inside of the 'if' statement so that the 'for' implementation can continue if the expression evaluation is false.
- Finally, we will go back to our label so that we can evaluate our 'if' statement again.

Following is an example of how to carry out the implementation of the classic 'for' loop statement.

Example

```
@echo off
SET /A i=1
:loop
IF %i%==5 GOTO END
echo The value of i is %i%
SET /a i=%i%+1
GOTO :LOOP
:END
```

Output

The above command produces the following output.

```
The value of i is 1

The value of i is 2

The value of i is 3

The value of i is 4
```

Looping through Command Line Arguments

The 'for' statement can also be used for checking command line arguments. The following example shows how the 'for' statement can be used to loop through the command line arguments.

Example

```
@ECHO OFF
:Loop
IF "%1"=="" GOTO completed
FOR %%F IN (%1) DO echo %%F
SHIFT
GOTO Loop
:completed
```

Output

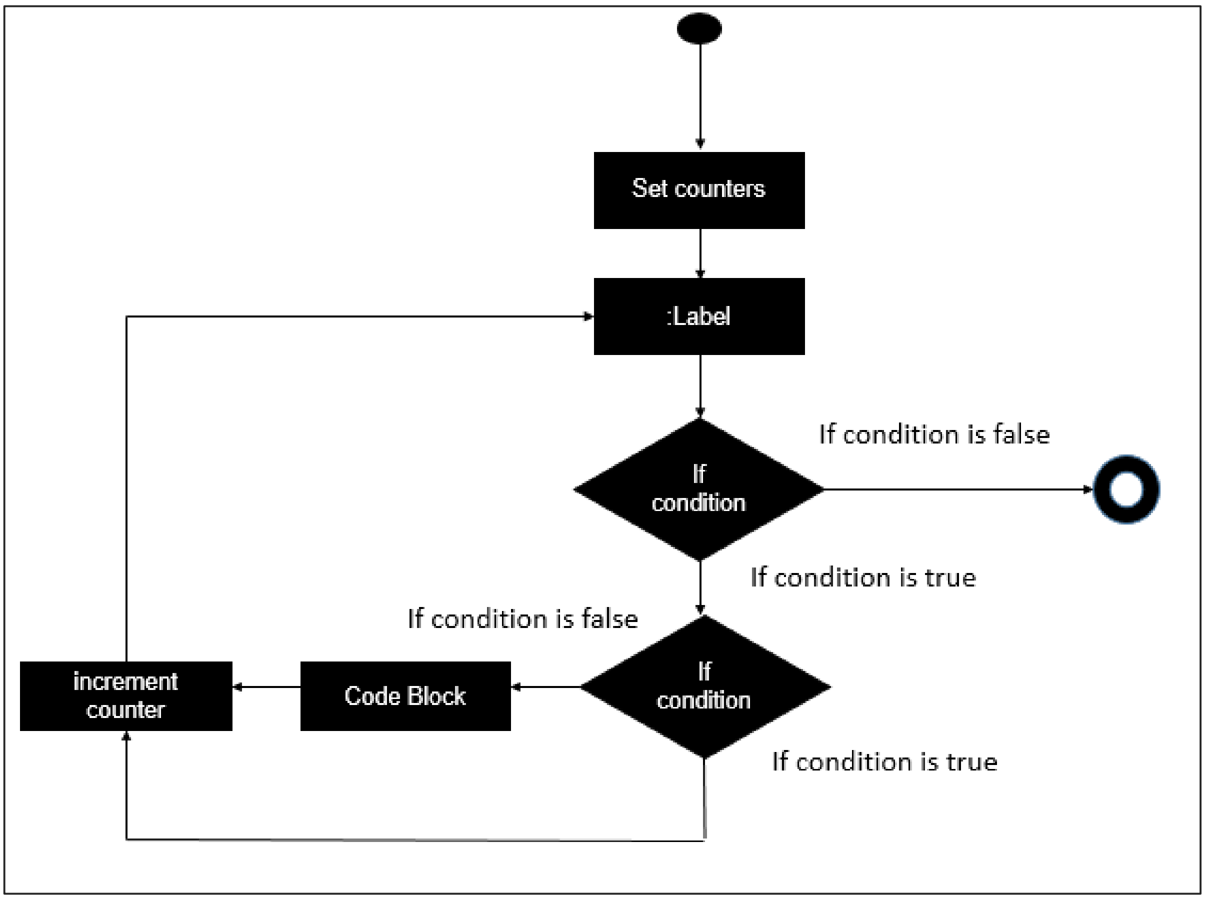
Let’s assume that our above code is stored in a file called Test.bat. The above command will produce the following output if the batch file passes the command line arguments of 1,2 and 3 as Test.bat 1 2 3.

```
1
2
3
```

Break Statement Implementation

The break statement is used to alter the flow of control inside loops within any programming language. The break statement is normally used in looping constructs and is used to cause immediate termination of the innermost enclosing loop.

The Batch Script language does not have a direct ‘for’ statement which does a break but this can be implemented by using labels. The following diagram shows the diagrammatic explanation of the break statement implementation in Batch Script.



The key thing to note about the above implementation is the involvement of two 'if' conditions. The second 'if' condition is used to control when the break is implemented. If the second 'if' condition is evaluated to be true, then the code block is not executed and the counter is directly implemented.

Following is an example of how to carry out the implementation of the break statement.

Example

```
@echo off
SET /A "index=1"
SET /A "count=5"
:while
if %index% leq %count% (
    if %index% == 2 goto :Increment
      echo The value of index is %index%
:Increment
    SET /A "index = index + 1"
    goto :while
)
```

The key thing to note about the above program is the addition of a label called :Increment. When the value of index reaches 2, we want to skip the statement which echoes its value to the command prompt and directly just increment the value of index.

Output

The above command produces the following output.

```
The value of index is 1
The value of index is 3
The value of index is 4
The value of index is 5
```

15. Batch Script – Functions

A function is a set of statements organized together to perform a specific task. In batch scripts, a similar approach is adopted to group logical statements together to form a function.

As like any other languages, functions in Batch Script follows the same procedure:

- **Function Declaration**: It tells the compiler about a function's name, return type, and parameters.

- **Function Definition**: It provides the actual body of the function.

Function Definition

In Batch Script, a function is defined by using the label statement. When a function is newly defined, it may take one or several values as input 'parameters' to the function, process the functions in the main body, and pass back the values to the functions as output 'return types'.

Every function has a function name, which describes the task that the function performs. To use a function, you "call" that function with its name and pass its input values (known as arguments) that matches the types of the function's parameters.

Following is the syntax of a simple function.

```
:function_name
Do_something
EXIT /B 0
```

- The function_name is the name given to the function which should have some meaning to match what the function actually does.

- The EXIT statement is used to ensure that the function exits properly.

Following is an example of a simple function.

Example

```
:Display
SET /A index=2
echo The value of index is %index%
EXIT /B 0
```

Calling a Function

A function is called in Batch Script by using the call command. Following is the syntax.

```
call :function_name
```

Following example shows how a function can be called from the main program.

Example

```
@echo off
SETLOCAL
CALL :Display
EXIT /B %ERRORLEVEL%
:Display
SET /A index=2
echo The value of index is %index%
EXIT /B 0
```

One key thing to note when defining the main program is to ensure that the statement EXIT /B %ERRORLEVEL% is put in the main program to separate the code of the main program from the function.

Output

The above command produces the following output.

```
The value of index is 2
```

Functions with Parameters

Functions can work with parameters by simply passing them when a call is made to the function.

Syntax

```
Call :function_name parameter1, parameter2… parametern
```

The parameters can then be accessed from within the function by using the tilde (~) character along with the positional number of the parameter.

Following example shows how a function can be called with parameters.

Example

```
@echo off

SETLOCAL

CALL :Display 5 , 10

EXIT /B %ERRORLEVEL%

:Display

echo The value of parameter 1 is %~1
echo The value of parameter 2 is %~2
EXIT /B 0
```

As seen in the above example, ~1 is used to access the first parameter sent to the function, similarly ~2 is used to access the second parameter.

Output

The above command produces the following output.

```
The value of parameter 1 is 5
The value of parameter 2 is 10
```

Functions with Return Values

Functions can work with return values by simply passing variables names which will hold the return values when a call is made to the function as shown below

Syntax

```
Call :function_name value1, value2… valuen
```

The return values are set in the function using the set command and the tilde(~) character along with the positional number of the parameter.

Following example shows how a function can be called with return values.

Example

```
@echo off
SETLOCAL
CALL :SetValue value1,value2
echo %value1%
echo %value2%
EXIT /B %ERRORLEVEL%
:SetValue
set "%~1=5"
set "%~2=10"
EXIT /B 0
```

Output

The above command produces the following output.

```
5
10
```

Local Variables in Functions

Local variables in functions can be used to avoid name conflicts and keep variable changes local to the function. The SETLOCAL command is first used to ensure the command processor takes a backup of all environment variables. The variables can be restored by calling ENDLOCAL command. Changes made in between are local to the current batch script. ENDLOCAL is automatically called when the end of the batch file is reached, i.e. by calling GOTO:EOF.

Localizing variables with SETLOCAL allows using variable names within a function freely without worrying about name conflicts with variables used outside the function.

Following example shows how local variables can be used in functions.

Example

```
@echo off
set str=Outer
echo %str%
CALL :SetValue str
echo %str%
EXIT /B %ERRORLEVEL%
:SetValue
SETLOCAL
set str=Inner
set "%~1=%str%"
ENDLOCAL
EXIT /B 0
```

Output

In the above program, the variable 'str' is being localized in the function SetValue. Thus even though the str value is being returned back to the main function, the value of str in the main function will not be replaced by the value being returned from the function.

The above command produces the following output.

```
Outer
Outer
```

Recursive Functions

The ability to completely encapsulate the body of a function by keeping variable changes local to the function and invisible to the caller. We can now have the ability to call a function recursively making sure each level of recursion works with its own set of variables even though variable names are being reused.

Following example shows how recursive functions can be used.

Example

The example shows how to calculate a Fibonacci number recursively. The recursion stops when the Fibonacci algorism reaches a number greater or equal to a given input number. The example starts with the numbers 0 and 1, the :myFibo function calls itself recursively to calculate the next Fibonacci number until it finds the Fibonacci number greater or equal to 1000000000.

The first argument of the myFibo function is the name of the variable to store the output in. This variable must be initialized to the Fibonacci number to start with and will be used as current Fibonacci number when calling the function and will be set to the subsequent Fibonacci number when the function returns.

```
@echo off
set "fst=0"
set "fib=1"
set "limit=1000000000"
call:myFibo fib,%fst%,%limit%
echo.The next Fibonacci number greater or equal %limit% is %fib%.
echo.&pause&goto:eof
:myFibo  -- calculate recursively
:myFibo  -- calculate recursively the next Fibonacci number greater or equal to
a limit
SETLOCAL
set /a "Number1=%~1"
set /a "Number2=%~2"
set /a "Limit=%~3"
set /a "NumberN=Number1 + Number2"
if /i %NumberN% LSS %Limit% call:myFibo NumberN,%Number1%,%Limit%
(ENDLOCAL
    IF "%~1" NEQ "" SET "%~1=%NumberN%"
)goto:eof
```

Output

The above command produces the following output.

```
The next Fibonacci number greater or equal 1000000000 is 1134903170.
```

File I/O

In Batch Script, it is possible to perform the normal file I/O operations that would be expected in any programming language.

Following are some of the operations that can performed on files.

- Creating files
- Reading files
- Writing to files
- Deleting files
- Moving files
- Renaming files

Creating Files

The creation of a new file is done with the help of the redirection filter >. This filter can be used to redirect any output to a file. Following is a simple example of how to create a file using the redirection command.

Example

```
@echo off
echo "Hello">C:\new.txt
```

Output

If the file new.txt is not present in C:\, then it will be created with the help of the above command.

Writing to Files

Content writing to files is also done with the help of the redirection filter >. This filter can be used to redirect any output to a file. Following is a simple example of how to create a file using the redirection command to write data to files.

Example

```
@echo off
dir C:\>C:\new.txt
```

The above code snippet first uses the DIR command to get the directory listing of the entire C:\ . It then takes that output and with the help of the redirection command sends it to the file new.txt.

Output

If you open up the file new.txt on your C drive, you will get the contents of your C drive in this file. Following is a sample output.

```

Volume in drive C is Windows8_OS
 Volume Serial Number is E41C-6F43

 Directory of C:\

12/22/2015  09:02 PM    <DIR>          01 - Music
06/14/2015  10:31 AM    <DIR>          02 - Videos
09/12/2015  06:23 AM    <DIR>          03 - Pictures
12/17/2015  12:19 AM    <DIR>          04 - Software
12/15/2015  11:06 PM    <DIR>          05 - Studies
12/20/2014  09:09 AM    <DIR>          06 - Future
12/20/2014  09:07 AM    <DIR>          07 - Fitness
09/19/2015  09:56 AM    <DIR>          08 - Tracking
10/19/2015  10:28 PM    <DIR>          09 – Misc
```

Appending to Files

Content writing to files is also done with the help of the double redirection filter >>. This filter can be used to append any output to a file. Following is a simple example of how to create a file using the redirection command to append data to files.

Example

```
@echo off
echo "This is the directory listing of C:\ Drive">C:\new.txt
dir C:\>>C:\new.txt
```

In the above example, you can see that the first echo command is used to create the file using the single redirection command whereas the DIR command is outputted to the file using the double redirection filter.

Output

If you open the file new.txt on your C drive, you will get the contents of your C drive in this file plus the string “This is the directory listing of C:\ Drive”. Following is a sample output.

```
"This is the directory listing of C:\ Drive"
Volume in drive C is Windows8_OS
 Volume Serial Number is E41C-6F43

 Directory of C:\

12/22/2015  09:02 PM    <DIR>          01 - Music
06/14/2015  10:31 AM    <DIR>          02 - Videos
09/12/2015  06:23 AM    <DIR>          03 - Pictures
12/17/2015  12:19 AM    <DIR>          04 - Software
12/15/2015  11:06 PM    <DIR>          05 - Studies
12/20/2014  09:09 AM    <DIR>          06 - Future
12/20/2014  09:07 AM    <DIR>          07 - Fitness
09/19/2015  09:56 AM    <DIR>          08 - Tracking
10/19/2015  10:28 PM    <DIR>          09 – Misc
```

Reading from Files

Reading of files in a batch script is done via using the FOR loop command to go through each line which is defined in the file that needs to be read. Since there is a no direct command to read text from a file into a variable, the 'for' loop needs to be used to serve this purpose.

Let's look at an example on how this can be achieved.

Example

```
@echo off
FOR /F "tokens=* delims=" %%x in (new.txt) DO echo %%x
```

The delims parameter is used to break up the text in the file into different tokens or words. Each word or token is then stored in the variable x. For each word which is read from the file, an echo is done to print the word to the console output.

Output

If you consider the new.txt file which has been considered in previous examples, you might get the following output when the above program is run.

```
"This is the directory listing of C:\ Drive"
Volume in drive C is Windows8_OS
 Volume Serial Number is E41C-6F43


 Directory of C:\


12/22/2015  09:02 PM    <DIR>          01 - Music
06/14/2015  10:31 AM    <DIR>          02 - Videos
09/12/2015  06:23 AM    <DIR>          03 - Pictures
12/17/2015  12:19 AM    <DIR>          04 - Software
12/15/2015  11:06 PM    <DIR>          05 - Studies
```

```
12/20/2014  09:09 AM    <DIR>          06 - Future
12/20/2014  09:07 AM    <DIR>          07 - Fitness
09/19/2015  09:56 AM    <DIR>          08 - Tracking
10/19/2015  10:28 PM    <DIR>          09 – Misc
```

Deleting Files

For deleting files, Batch Script provides the DEL command.

Syntax

```
DEL [/P] [/F] [/S] [/Q] [/A[[:]attributes]] names
```

Following are the description of the options which can be presented to the DEL command.

Names	**Specifies a list of one or more files or directories. Wildcards may be used to delete multiple files. If a directory is specified, all files within the directory will be deleted**
/P	Prompts for confirmation before deleting each file.
/F	Force deletes of read-only files.
/S	Deletes specified files from all subdirectories.
/Q	Quiet mode, do not ask if ok to delete on global wildcard.
/A	Selects files to delete based on attributes.
attributes	R Read-only files, S System files, H Hidden files, A Files ready for archiving - Prefix meaning not

Following examples show how the DEL command can be used.

Examples

```
del test.bat
```

The above command will delete the file test.bat in the current directory, if the file exists.

```
del c:\test.bat
```

The above command will delete the file C:\test.bat in the current directory, if the file exists.

```
del c:\*.bat
```

The * (asterisks) is a wild character. *.bat indicates that you would like to delete all bat files in the c:\directory.

```
del c:\?est.tmp
```

The ? (question mark) is a single wild character for one letter. The use of this command in the above example will delete any file ending with "est.tmp", such as pest.tmp or test.tmp.

Renaming Files

For renaming files, Batch Script provides the REN or RENAME command.

Syntax

```
RENAME [drive:][path][directoryname1 | filename1] [directoryname2 | filename2]
```

Let’s take a look at some examples of renaming files.

Examples

```
rename *.txt *.bak
```

The above command will rename all text files to files with .bak extension.

```
rename "TESTA.txt" "TESTB.txt"
```

The above command will rename the file TESTA.txt to TESTB.txt.

Moving Files

For moving files, Batch Script provides the MOVE command.

Syntax

```
MOVE [/Y | /-Y] [drive:][path]filename1[,...] destination
```

Following are the description of the options which can be presented to the DEL command.

[drive:][path]filename1	**Specifies the location and name of the file or files you want to move**
destination	Specifies the new location of the file. Destination can consist of a drive letter and colon, a directory name, or a combination. If you are moving only one file, you can also include a filename if you want to rename the file when you move it.
[drive:][path]dirname1	Specifies the directory you want to rename.
dirname2	Specifies the new name of the directory.
/Y	Suppresses prompting to confirm you want to overwrite an existing destination file.
/-Y	Causes prompting to confirm you want to overwrite an existing destination file.

Let’s look at some examples of renaming files.

Examples

```
move c:\windows\temp\*.* c:\temp
```

The above command will move the files of c:\windows\temp to the temp directory in root.

```
move new.txt, test.txt c:\example
```

The above command will move the files new.txt and test.txt into the c:\example folder.

Batch Files – Pipes

The pipe operator (|) takes the output (by default, STDOUT) of one command and directs it into the input (by default, STDIN) of another command. For example, the following command sorts the contents of the directory C:\

```
dir C:\ | sort
```

In this example, both commands start simultaneously, but then the sort command pauses until it receives the dir command's output. The sort command uses the dir command's output as its input, and then sends its output to handle 1 (that is, STDOUT).

Following is another example of the pipe command. In this example, the contents of the file C:\new.txt are sent to the sort command through the pipe filter.

```
@echo off
TYPE C:\new.txt | sort
```

Combining Commands with Redirection Operators

Usually, the pipe operator is used along with the redirection operator to provide useful functionality when it comes to working with pipe commands.

For example, the below command will first take all the files defined in C:\, then using the pipe command, will find all the files with the .txt extension. It will then take this output and print it to the file AllText.txt.

```
dir C:\ | find "txt" > AllText.txt
```

Using Multiple Pipe Commands

To use more than one filter in the same command, separate the filters with a pipe (|). For example, the following command searches every directory on drive C:, finds the file names that include the string "Log", and then displays them in one Command Prompt window at a time:

```
dir c:\ /s /b | find "TXT" | more
```

Following are some examples of how the pipe filter can be used.

Examples

The following example send's the list of all running tasks using the tasklist command and sends the output to the find command. The find command will then find all processes which are of the type notepad and display them in the command prompt.

```
tasklist | find "notepad"
```

Output

Following is a sample output.

```
notepad.exe                  1400 Console                    1      8,916 K
notepad.exe                  4016 Console                    1     11,200 K
notepad.exe                  1508 Console                    1      8,720 K
notepad.exe                  4076 Console                    1      8,688 K
```

The following example send's the list of all running tasks using the tasklist command and sends the output to the more command. The more command will then display the lists of running tasks one page at a time.

```
tasklist | more
```

Output

```
Image Name                     PID Session Name        Session#    Mem Usage
========================= ======== ================ =========== ============
System Idle Process              0 Services                   0          4 K
System                           4 Services                   0        276 K
smss.exe                       344 Services                   0      1,060 K
csrss.exe                      524 Services                   0      4,188 K
csrss.exe                      608 Console                    1     58,080 K
wininit.exe                    616 Services                   0      3,528 K
winlogon.exe                   644 Console                    1      5,636 K
services.exe                   708 Services                   0      7,072 K
lsass.exe                      716 Services                   0     10,228 K
svchost.exe                    784 Services                   0     10,208 K
svchost.exe                    828 Services                   0      7,872 K
dwm.exe                        912 Console                    1    208,316 K
nvvsvc.exe                     932 Services                   0      6,772 K
nvxdsync.exe                   968 Console                    1     16,584 K
nvvsvc.exe                     976 Console                    1     12,780 K
svchost.exe                   1008 Services                   0     20,340 K
svchost.exe                    224 Services                   0     39,740 K
svchost.exe                    468 Services                   0     11,864 K
svchost.exe                    860 Services                   0     11,184 K
svchost.exe                    232 Services                   0     16,992 K
wlanext.exe                   1168 Services                   0     12,840 K
-- More  --
```

The following example send's the list of all running tasks using the tasklist command and sends the output to the find command. The find command will then find all processes which are of the type notepad and then uses the redirection command to send the content to the file tasklist.txt.

```
tasklist | find "notepad" > tasklist.txt
```

Output

If you open the file tasklist.txt, you will get the following sample output.

```
notepad.exe                   1400 Console                    1      8,916 K
notepad.exe                   4016 Console                    1     11,200 K
notepad.exe                   1508 Console                    1      8,720 K
notepad.exe                   4076 Console                    1      8,688 K
```

Batch Files – Inputs

When a batch file is run, it gives you the option to pass in command line parameters which can then be read within the program for further processing. The batch files parameters can be recalled from within the program using the % operator along with the numeric position of the parameter. Following is how the command line parameters are defined.

- %0 is the program name as it was called.
- %1 is the first command line parameter.
- %2 is the second command line parameter.
- So on till %9.

Let's take a look at a simple example of how command line parameters can be used.

Example

```
@echo off
echo The first parameter is %1
echo The second parameter is %2
echo The third parameter is %3
```

Output

If the above code is stored in a file called test.bat and the file is run as

```
test.bat 5 10 15
```

then, following will be the output.

```
The first parameter is 5
The second parameter is 10
The third parameter is 15
```

Using the SHIFT Operator

One of the limitations of command line arguments is that it can accept only arguments till %9. Let’s take an example of this limitation.

Example

```
@echo off
echo %1
echo %2
echo %3
echo %4
echo %5
echo %6
echo %7
echo %8
echo %9
echo %10
```

Output

If the above code is stored in a file called test.bat and the file is run as

```
test.bat a b c d e f g h i j
```

Then following will be the output.

```
a
b
c
d
e
f
```

```
h
i
a0
```

As you can see from the above output, the final value which should be shown as 'j' is being shown as a0. This is because there is no parameter known as %10.

This limitation can be avoided by using the SHIFT operator. After your batch file handled its first parameter(s) it could SHIFT them (just insert a line with only the command SHIFT), resulting in %1 getting the value B, %2 getting the value C, etcetera, till %9, which now gets the value J. Continue this process until at least %9 is empty.

Let's look at an example of how to use the SHIFT operator to overcome the limitation of command line arguments.

Example

```
@ECHO OFF
:Loop
IF "%1"=="" GOTO Continue
   echo %1%
SHIFT
GOTO Loop
:Continue
```

If the above code is stored in a file called test.bat and the file is run as

```
test.bat a b c d e f g h i j
```

Then following will be the output.

```
a
b
c
d
e
f
h
i
j
```

Note

Some characters in the command line are ignored by batch files, depending on the DOS version, whether they are "escaped" or not, and often depending on their location in the command line:

- Commas (",") are replaced by spaces, unless they are part of a string in doublequotes.
- Semicolons (";") are replaced by spaces, unless they are part of a string in doublequotes.
- "=" characters are sometimes replaced by spaces, not if they are part of a string in doublequotes.
- The first forward slash ("/") is replaced by a space only if it immediately follows the command, without a leading space.
- Multiple spaces are replaced by a single space, unless they are part of a string in doublequotes.
- Tabs are replaced by a single space.
- Leading spaces before the first command line argument are ignored.
- Trailing spaces after the last command line argument are trimmed.

Folders

In Batch Script, it is possible to perform the normal folder based operations that would be expected in any programming language.

Following are some of the operations that can be performed on folders.

- Creating folders
- Listing folders
- Traversing files in folders
- Deleting folders
- Renaming folders

Creating Folders

The creation of a folder is done with the assistance of the MD (Make directory) command.

Syntax

```
MKDIR [drive:]path
MD [drive:]path
```

Let's look at some examples on how to use the MD command.

Examples

```
md test
```

The above command will create a directory called test in your current directory.

```
md C:\test
```

The above command will create a directory called test in the C drive.

```
md “Test A”
```

If there are spaces in the folder name, then the folder name should be given in quotes.

```
mkdir \a\b\c
```

The above command creates directories recursively and is the same as issuing the following set of commands.

```
mkdir \a
chdir \a
mkdir b
chdir b
mkdir c
```

Listing Folder Contents

The listing of folder contents can be done with the dir command. This command allows you to see the available files and directories in the current directory. The dir command also shows the last modification date and time, as well as the file size.

Syntax

```
DIR [drive:][path][filename] [/A[[:]attributes]] [/B] [/C] [/D] [/L] [/N]
[/O[[:]sortorder]] [/P] [/Q] [/R] [/S] [/T[[:]timefield]] [/W] [/X] [/4]
```

[drive:][path][filename]	**Specifies drive, directory, or files to list**
/A	Displays files with specified attributes.
attributes	D - Directories R - Read-only files H - Hidden files A - Files ready for archiving S - System files I - Not content indexed files L - Reparse Points - Prefix meaning not
/B	Uses bare format (no heading information or summary).
/C	Displays the thousand separator in file sizes. This is the default. Use /-C to disable the display of the separator.
/D	Same as wide but files are list sorted by column.
/L	Uses lowercase.
/N	New long list format where filenames are on the far right.
/O	Lists by files in sorted order.
sortorder	N By name (alphabetic), S By size (smallest first), E By extension (alphabetic), D By date/time (oldest first), G Group directories first - Prefix to reverse order
/P	Pauses after each screen is full of information.
/Q	Displays the owner of the file.
/R	Displays alternate data streams of the file.
/S	Displays files in the specified directory and all subdirectories.
/T	Controls what time field is displayed or used for sorting.
timefield	C - Creation A - Last Access W - Last Written
/W	Uses wide list format.
/X	This displays the short names generated for non-8dot3 file names. The format is that of /N with the short name inserted before the long name. If no short name is present, blanks are displayed in its place.
/4	Displays four-digit years.

Let's see some of the examples on how to use the DIR command.

Examples

```
dir *.exe
```

The above command lists any file that ends with the .exe file extension.

```
dir *.txt *.doc
```

The above command uses multiple filespecs to list any files ending with .txt and .doc in one command.

```
dir /ad
```

Lists only the directories in the current directory. If you need to move into one of the directories listed use the cd command.

```
dir /s
```

Lists the files in the directory that you are in and all sub directories after that directory. If you are at root "C:\>", type this command, this will list to you every file and directory on the C: drive of the computer.

```
dir /p
```

If the directory has lots of files and you cannot read all the files as they scroll by, you can use the above command and it displays all files one page at a time.

```
dir /w
```

If you don't need file information you can use the above command to list only the files and directories going horizontally, taking as little space as needed.

```
dir /s /w /p
```

The above command will list all the files and directories in the current directory and the sub directories, in wide format and one page at a time.

Deleting Folders

For deleting folders, Batch Script provides the DEL command.

Syntax

```
DEL [/P] [/F] [/S] [/Q] [/A[[:]attributes]] names
```

Following are the description of the options which can be presented to the DEL command.

Names	Specifies a list of one or more files or directories. Wildcards may be used to delete multiple files. If a directory is specified, all files within the directory will be deleted
/P	Prompts for confirmation before deleting each file.
/F	Force deletes read-only files.
/S	Deletes specified files from all subdirectories.
/Q	Quiet mode, do not ask if ok to delete on global wildcard.
/A	Selects files to delete based on attributes.
attributes	R - Read-only files, S - System files, H - Hidden files, A - Files ready for archiving - Prefix meaning not

Let's look at some examples of how the DEL command can be used for folders.

Examples

```
del Example
```

The above command will delete the folder called Example in the current working directory.

```
del C:\Example
```

The above command will delete the folder called Example in C drive.

```
Del Example1 , Example2
```

The above command will delete the folder called Example1 and Example2 in the current working directory.

Renaming Folders

For renaming folders, Batch Script provides the REN or RENAME command.

Syntax

```
RENAME [drive:][path][directoryname1 | filename1] [directoryname2 | filename2]
```

Let's look at some examples of renaming folders.

Examples

```
ren Example Example1
```

The above command will rename the folder called Example in the current working directory to Example1.

```
del C:\Example Example1
```

The above command will rename the folder called Example in C Drive to Example1.

Moving Folders

For moving folders, Batch Script provides the MOVE command.

Syntax

```
MOVE [/Y | /-Y] [drive:][path]filename1[,...] destination
```

Following are the description of the options which can be presented to the DEL command.

[drive:][path]filename1	**Specifies the location and name of the file or files you want to move**
destination	Specifies the new location of the file. Destination can consist of a drive letter and colon, a directory name, or a combination. If you are moving only one file, you can also include a filename if you want to rename the file when you move it.
[drive:][path]dirname1	Specifies the directory you want to rename.
dirname2	Specifies the new name of the directory.
/Y	Suppresses prompting to confirm you want to overwrite an existing destination file.
/-Y	Causes prompting to confirm you want to overwrite an existing destination file.

Let's look at some examples of moving folders.

Examples

```
move *.* C:\Example
```

The above command will move all files from the current directory to the folder C:\Example.

```
move *.txt C:\Example
```

The above command will move all files with the txt extension from the current directory to the folder C:\Example.

```
move C:\old\*.* C:\Example
```

The above command will move all files from the folder called 'old' in C drive to the folder C:\Example.

16. Batch Script – Process

In this chapter, we will discuss the various processes involved in Batch Script.

Viewing the List of Running Processes

In Batch Script, the TASKLIST command can be used to get the list of currently running processes within a system.

Syntax

```
TASKLIST [/S system [/U username [/P [password]]]] [/M [module] | /SVC | /V]
[/FI filter] [/FO format] [/NH]
```

Following are the description of the options which can be presented to the TASKLIST command.

/S system	Specifies the remote system to connect to
/U [domain\]user	Specifies the user context under which the command should execute.
/P [password]	Specifies the password for the given user context. Prompts for input if omitted.
/M [module]	Lists all tasks currently using the given exe/dll name. If the module name is not specified all loaded modules are displayed.
/SVC	Displays services hosted in each process.
/V	Displays verbose task information.
/FI filter	Displays a set of tasks that match a given criteria specified by the filter.
/FO format	Specifies the output format. Valid values: "TABLE", "LIST", "CSV".
/NH	Specifies that the "Column Header" should not show in the output. Valid only for "TABLE" and "CSV" formats.

Examples

```
TASKLIST
```

The above command will get the list of all the processes running on your local system. Following is a snapshot of the output which is rendered when the above command is run as it is. As you can see from the following output, not only do you get the various processes running on your system, you also get the memory usage of each process.

```
Image Name                     PID Session Name        Session#    Mem Usage
========================= ======== ================ =========== ============
System Idle Process              0 Services                   0          4 K
System                           4 Services                   0        272 K
smss.exe                       344 Services                   0      1,040 K
csrss.exe                      528 Services                   0      3,892 K
csrss.exe                      612 Console                    1     41,788 K
wininit.exe                    620 Services                   0      3,528 K
winlogon.exe                   648 Console                    1      5,884 K
services.exe                   712 Services                   0      6,224 K
lsass.exe                      720 Services                   0      9,712 K
svchost.exe                    788 Services                   0     10,048 K
svchost.exe                    832 Services                   0      7,696 K
dwm.exe                        916 Console                    1    117,440 K
nvvsvc.exe                     932 Services                   0      6,692 K
nvxdsync.exe                   968 Console                    1     16,328 K
nvvsvc.exe                     976 Console                    1     12,756 K
svchost.exe                   1012 Services                   0     21,648 K
svchost.exe                    236 Services                   0     33,864 K
svchost.exe                    480 Services                   0     11,152 K
svchost.exe                   1028 Services                   0     11,104 K
svchost.exe                   1048 Services                   0     16,108 K
wlanext.exe                   1220 Services                   0     12,560 K
conhost.exe                   1228 Services                   0      2,588 K
svchost.exe                   1276 Services                   0     13,888 K
svchost.exe                   1420 Services                   0     13,488 K
spoolsv.exe                   1556 Services                   0      9,340 K
```

```
tasklist > process.txt
```

The above command takes the output displayed by tasklist and saves it to the process.txt file.

```
tasklist /fi "memusage gt 40000"
```

The above command will only fetch those processes whose memory is greater than 40MB. Following is a sample output that can be rendered.

```
Image Name                     PID Session Name        Session#    Mem Usage
========================= ======== ================ =========== ============
dwm.exe                        916 Console                    1    127,912 K
explorer.exe                  2904 Console                    1    125,868 K
ServerManager.exe             1836 Console                    1     59,796 K
WINWORD.EXE                   2456 Console                    1    144,504 K
chrome.exe                    4892 Console                    1    123,232 K
chrome.exe                    4976 Console                    1     69,412 K
chrome.exe                    1724 Console                    1     76,416 K
chrome.exe                    3992 Console                    1     56,156 K
chrome.exe                    1168 Console                    1    233,628 K
chrome.exe                     816 Console                    1     66,808 K
```

Killing a Particular Process

Allows a user running Microsoft Windows XP professional, Windows 2003, or later to kill a task from a Windows command line by process id (PID) or image name. The command used for this purpose is the TASKILL command.

Syntax

```
TASKKILL [/S system [/U username [/P [password]]]] { [/FI filter] [/PID
processid | /IM imagename] } [/T] [/F]
```

Following are the description of the options which can be presented to the TASKKILL command.

/S system	Specifies the remote system to connect to
/U [domain\]user	Specifies the user context under which the command should execute.
/P [password]	Specifies the password for the given user context. Prompts for input if omitted.
/FI FilterName	Applies a filter to select a set of tasks. Allows "*" to be used. ex. imagename eq acme* See below filters for additional information and examples.
/PID processID	Specifies the PID of the process to be terminated. Use TaskList to get the PID.
/IM ImageName	Specifies the image name of the process to be terminated. Wildcard '*' can be used to specify all tasks or image names.
/T	Terminates the specified process and any child processes which were started by it.
/F	Specifies to forcefully terminate the process(es).

Examples

```
taskkill /f /im notepad.exe
```

The above command kills the open notepad task, if open.

```
taskill /pid 9214
```

The above command kills a process which has a process of 9214.

Starting a New Process

DOS scripting also has the availability to start a new process altogether. This is achieved by using the START command.

Syntax

```
START "title" [/D path] [options] "command" [parameters]
```

Wherein

- title - Text for the CMD window title bar (required.)
- path - Starting directory.
- command - The command, batch file or executable program to run.
- parameters - The parameters passed to the command.

Following are the description of the options which can be presented to the START command.

/MIN	Start window Minimized
/MAX	Start window maximized.
/LOW	Use IDLE priority class.
/NORMAL	Use NORMAL priority class.
/ABOVENORMAL	Use ABOVENORMAL priority class.
/BELOWNORMAL	Use BELOWNORMAL priority class.
/HIGH	Use HIGH priority class.
/REALTIME	Use REALTIME priority class.

Examples

```
START "Test Batch Script" /Min test.bat
```

The above command will run the batch script test.bat in a new window. The windows will start in the minimized mode and also have the title of "Test Batch Script".

```
START "" "C:\Program Files\Microsoft Office\Winword.exe" "D:\test\TESTA.txt"
```

The above command will actually run Microsoft word in another process and then open the file TESTA.txt in MS Word.

17. Batch Script – Aliases

Aliases means creating shortcuts or keywords for existing commands. Suppose if we wanted to execute the below command which is nothing but the directory listing command with the /w option to not show all of the necessary details in a directory listing.

```
Dir /w
```

Suppose if we were to create a shortcut to this command as follows.

```
dw= dir /w
```

When we want to execute the **dir /w** command, we can simply type in the word **dw**. The word 'dw' has now become an alias to the command Dir /w.

Creating an Alias

Alias are managed by using the **doskey** command.

Syntax

```
DOSKEY [options] [macroname=[text]]
```

Wherein

- **macroname** - A short name for the macro.
- **text** - The commands you want to recall.

Following are the description of the options which can be presented to the DOSKEY command.

/REINSTALL	Installs a new copy of Doskey
/LISTSIZE=size	Sets size of command history buffer.
/MACROS	Displays all Doskey macros.
/MACROS:ALL	Displays all Doskey macros for all executables which have Doskey macros.
/MACROS:exename	Displays all Doskey macros for the given executable.
/HISTORY	Displays all commands stored in memory.
/INSERT	Specifies that new text you type is inserted in old text.
/OVERSTRIKE	Specifies that new text overwrites old text.
/EXENAME=exename	Specifies the executable.
/MACROFILE=filename	Specifies a file of macros to install.

macroname	Specifies a name for a macro you create.
text	Specifies commands you want to record.

Example

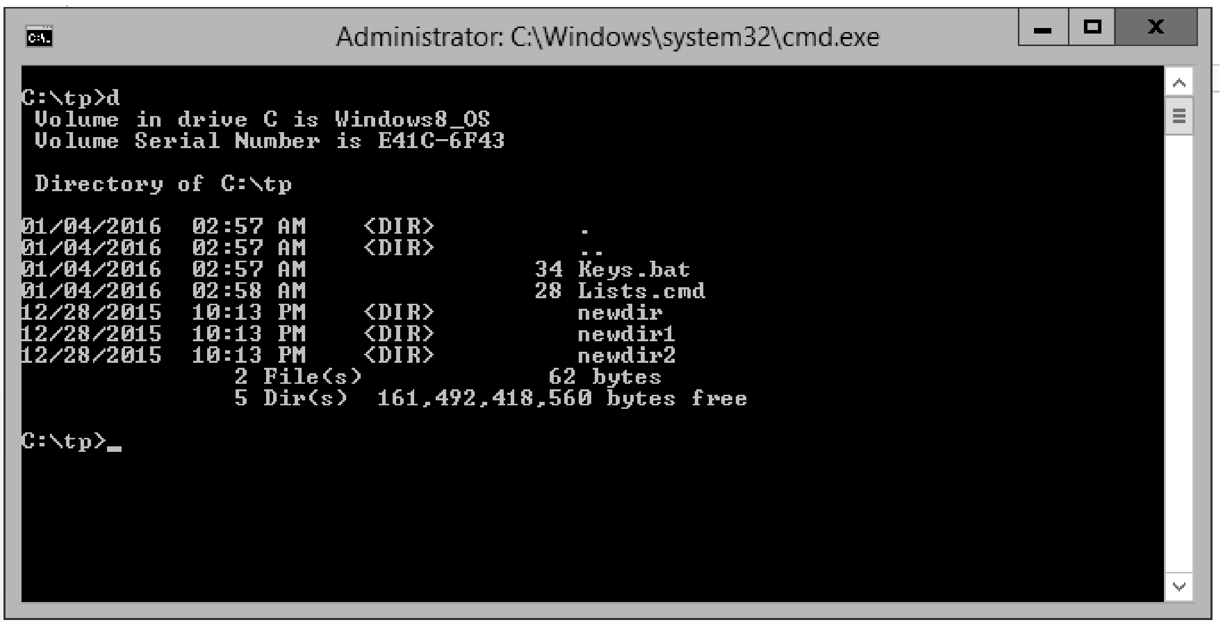
Create a new file called keys.bat and enter the following commands in the file. The below commands creates two aliases, one if for the cd command, which automatically goes to the directory called test. And the other is for the dir command.

```
@echo off
doskey cd=cd/test
doskey d=dir
```

Once you execute the command, you will able to run these aliases in the command prompt.

Output

The following screenshot shows that after the above created batch file is executed, you can freely enter the 'd' command and it will give you the directory listing which means that your alias has been created.



```
C:\tp>d
 Volume in drive C is Windows8_OS
 Volume Serial Number is E41C-6F43

 Directory of C:\tp

01/04/2016  02:57 AM    <DIR>          .
01/04/2016  02:57 AM    <DIR>          ..
01/04/2016  02:57 AM                34 Keys.bat
01/04/2016  02:58 AM                28 Lists.cmd
12/28/2015  10:13 PM    <DIR>          newdir
12/28/2015  10:13 PM    <DIR>          newdir1
12/28/2015  10:13 PM    <DIR>          newdir2
               2 File(s)             62 bytes
               5 Dir(s)  161,492,418,560 bytes free

C:\tp>_
```

Deleting an Alias

An alias or macro can be deleted by setting the value of the macro to NULL.

Example

```
@echo off
doskey cd=cd/test
doskey d=dir
d=
```

In the above example, we are first setting the macro d to d=dir. After which we are setting it to NULL. Because we have set the value of d to NULL, the macro d will deleted.

Replacing an Alias

An alias or macro can be replaced by setting the value of the macro to the new desired value.

Example

```
@echo off
doskey cd=cd/test
doskey d=dir

d=dir /w
```

In the above example, we are first setting the macro d to d=dir. After which we are setting it to dir /w. Since we have set the value of d to a new value, the alias 'd' will now take on the new value.

18. Batch Script – Devices

Windows now has an improved library which can be used in batch scripts for working with devices attached to the system. This is known as the device console – DevCon.exe.

Windows driver developers and testers can use DevCon to verify that a driver is installed and configured correctly, including the proper INF files, driver stack, driver files, and driver package. You can also use the DevCon commands (enable, disable, install, start, stop, and continue) in scripts to test the driver. **DevCon** is a command-line tool that performs device management functions on local computers and remote computers.

Display driver and device info DevCon can display the following properties of drivers and devices on local computers, and remote computers (running Windows XP and earlier):

- Hardware IDs, compatible IDs, and device instance IDs. These identifiers are described in detail in device identification strings.
- Device setup classes.
- The devices in a device setup class.
- INF files and device driver files.
- Details of driver packages.
- Hardware resources.
- Device status.
- Expected driver stack.
- Third-party driver packages in the driver store.
- Search for devices DevCon can search for installed and uninstalled devices on a local or remote computer by hardware ID, device instance ID, or device setup class.
- Change device settings DevCon can change the status or configuration of Plug and Play (PnP) devices on the local computer in the following ways:
    - Enable a device.
    - Disable a device.
    - Update drivers (interactive and non-interactive).
    - Install a device (create a devnode and install software).
    - Remove a device from the device tree and delete its device stack.
    - Rescan for Plug and Play devices.
    - Add, delete, and reorder the hardware IDs of root-enumerated devices.
    - Change the upper and lower filter drivers for a device setup class.
    - Add and delete third-party driver packages from the driver store.

DevCon (DevCon.exe) is included when you install the WDK, Visual Studio, and the Windows SDK for desktop apps. DevCon.exe kit is available in the following locations when installed.

```
%WindowsSdkDir%\tools\x64\devcon.exe

%WindowsSdkDir%\tools\x86\devcon.exe

%WindowsSdkDir%\tools\arm\devcon.exe
```

Syntax

```
devcon [/m:\\computer] [/r] command [arguments]
```

wherein

- **/m:\\computer** - Runs the command on the specified remote computer. The backslashes are required.

- **/r** - Conditional reboot. Reboots the system after completing an operation only if a reboot is required to make a change effective.

- **command** - Specifies a DevCon command.

- To list and display information about devices on the computer, use the following commands:
  - DevCon HwIDs
  - DevCon Classes
  - DevCon ListClass
  - DevCon DriverFiles
  - DevCon DriverNodes
  - DevCon Resources
  - DevCon Stack
  - DevCon Status
  - DevCon Dp_enum
- To search for information about devices on the computer, use the following commands:
  - DevCon Find
  - DevCon FindAll
- To manipulate the device or change its configuration, use the following commands:
  - DevCon Enable
  - DevCon Disable
  - DevCon Update
  - DevCon UpdateNI
  - DevCon Install
  - DevCon Remove
  - DevCon Rescan

- DevCon Restart
- DevCon Reboot
- DevCon SetHwID
- DevCon ClassFilter
- DevCon Dp_add
- DevCon Dp_delete

Examples

Following are some examples on how the DevCon command is used.

```
List all driver files
```

The following command uses the DevCon DriverFiles operation to list the file names of drivers that devices on the system use. The command uses the wildcard character (*) to indicate all devices on the system. Because the output is extensive, the command uses the redirection character (>) to redirect the output to a reference file, driverfiles.txt.

```
devcon driverfiles * > driverfiles.txt
```

The following command uses the DevCon status operation to find the status of all devices on the local computer. It then saves the status in the status.txt file for logging or later review. The command uses the wildcard character (*) to represent all devices and the redirection character (>) to redirect the output to the status.txt file.

```
devcon status * > status.txt
```

The following command enables all printer devices on the computer by specifying the Printer setup class in a DevCon Enable command. The command includes the /r parameter, which reboots the system if it is necessary to make the enabling effective.

```
devcon /r enable =Printer
```

The following command uses the DevCon Install operation to install a keyboard device on the local computer. The command includes the full path to the INF file for the device (keyboard.inf) and a hardware ID (*PNP030b).

```
devcon /r install c:\windows\inf\keyboard.inf *PNP030b
```

The following command will scan the computer for new devices.

```
devcon scan
```

The following command will rescan the computer for new devices.

```
devcon rescan
```

19. Batch Script – Registry

The Registry is one of the key elements on a windows system. It contains a lot of information on various aspects of the operating system. Almost all applications installed on a windows system interact with the registry in some form or the other.

The Registry contains two basic elements: keys and values. **Registry keys** are container objects similar to folders. **Registry values** are non-container objects similar to files. Keys may contain values or further keys. Keys are referenced with a syntax similar to Windows' path names, using backslashes to indicate levels of hierarchy.

This chapter looks at various functions such as querying values, adding, deleting and editing values from the registry.

Reading from the Registry

Reading from the registry is done via the REG QUERY command. This command can be used to retrieve values of any key from within the registry.

Syntax

```
REG QUERY [ROOT\]RegKey /v ValueName [/s]
REG QUERY [ROOT\]RegKey /ve  --This returns the (default) value
```

Where RegKey is the key which needs to be searched for in the registry.

Example

```
@echo off
REG QUERY HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Windows\
```

The above command will query all the keys and their respective values under the registry key HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Windows\

Output

The output will display all the keys and values under the registry key.

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Windows\
```

This location in the registry has some key information about the windows system such as the System Directory location.

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Windows
    Directory    REG_EXPAND_SZ    %SystemRoot%
    SystemDirectory    REG_EXPAND_SZ    %SystemRoot%\system32
    NoInteractiveServices    REG_DWORD    0x1
    CSDBuildNumber    REG_DWORD    0x4000
    ShellErrorMode    REG_DWORD    0x1
    ComponentizedBuild    REG_DWORD    0x1
    CSDVersion    REG_DWORD    0x0
    ErrorMode    REG_DWORD    0x0
    CSDReleaseType    REG_DWORD    0x0
    ShutdownTime    REG_BINARY    3AFEF5D05D46D101
```

Adding to the Registry

Adding to the registry is done via the REG ADD command. Note that in order to add values to the registry you need to have sufficient privileges on the system to perform this operation.

Syntax

The REG ADD command has the following variations. In the second variation, no name is specified for the key and it will add the name of "(Default)" for the key.

```
REG ADD [ROOT\]RegKey /v ValueName [/t DataType] [/S Separator] [/d Data] [/f]
REG ADD [ROOT\]RegKey /ve [/d Data] [/f]
```

Where

- **ValueName** - The value, under the selected RegKey, to edit.
- **/d Data** - The actual data to store as a "String", integer, etc.
- **/f** - Force an update without prompting "Value exists, overwrite Y/N".
- **/S Separator** - Character to use as the separator in REG_MULTI_SZ values. The default is "\0".
- **/t DataType** – These are the data types defined as per the registry standards which can be:
  - REG_SZ (default)
  - REG_DWORD
  - REG_EXPAND_SZ
  - REG_MULTI_SZ

Example

```
@echo off
REG ADD HKEY_CURRENT_USER\Console /v Test /d "Test Data"
REG QUERY HKEY_CURRENT_USER\Console /v Test
```

In the above example, the first part is to add a key into the registry under the location HKEY_CURRENT_USER\Console. This key will have a name of Test and the value assigned to the key will be Test Data which will be of the default string type.

The second command just displays what was added to the registry by using the REG QUERY command.

Output

Following will be the output of the above program. The first line of the output shows that the 'Add' functionality was successful and the second output shows the inserted value into the registry.

```
The operation completed successfully.
HKEY_CURRENT_USER\Console

    Test    REG_SZ    Test Data
```

Deleting from the Registry

Deleting from the registry is done via the REG DEL command. Note that in order to delete values from the registry you need to have sufficient privileges on the system to perform this operation.

Syntax

The REG DELETE command has the following variations. In the second variation, the default value will be removed and in the last variation all the values under the specified key will be removed.

```
REG DELETE [ROOT\]RegKey /v ValueName [/f]

   REG DELETE [ROOT\]RegKey /ve [/f]

   REG DELETE [ROOT\]RegKey /va [/f]
```

Where

- ValueName - The value, under the selected RegKey, to edit.
- /f - Force an update without prompting "Value exists, overwrite Y/N".

Example

```
@echo off

REG DELETE HKEY_CURRENT_USER\Console /v Test /f

REG QUERY HKEY_CURRENT_USER\Console /v Test
```

In the above example, the first part is to delete a key into the registry under the location HKEY_CURRENT_USER\Console. This key has the name of Test. The second command just displays what was deleted to the registry by using the REG QUERY command. From this command, we should expect an error, just to ensure that our key was in fact deleted.

Output

Following will be the output of the above program. The first line of the output shows that the 'Delete' functionality was successful and the second output shows an error which was expected to confirm that indeed our key was deleted from the registry.

```
The operation completed successfully.
ERROR: The system was unable to find the specified registry key or value.
```

Copying Registry Keys

Copying from the registry is done via the REG COPY command. Note that in order to copy values from the registry, you need to have sufficient privileges on the system to perform this operation on both the source location and the destination location.

Syntax

```
REG COPY  [\\SourceMachine\][ROOT\]RegKey [\\DestMachine\][ROOT\]RegKey
```

Example

```
@echo off
REG COPY HKEY_CURRENT_USER\Console HKEY_CURRENT_USER\Console\Test
REG QUERY HKEY_CURRENT_USER\Console\Test
```

In the above example, the first part is to copy the contents from the location HKEY_CURRENT_USER\Console into the location HKEY_CURRENT_USER\Console\Test on the same machine. The second command is used to query the new location to check if all the values were copied properly.

Output

Following is the output of the above program. The first line of the output shows that the 'Copy' functionality was successful and the second output shows the values in our copied location.

```
The operation completed successfully.
HKEY_CURRENT_USER\Console\Test
    HistoryNoDup    REG_DWORD    0x0
    FullScreen    REG_DWORD    0x0
    ScrollScale    REG_DWORD    0x1
    ExtendedEditKeyCustom    REG_DWORD    0x0
    CursorSize    REG_DWORD    0x19
    FontFamily    REG_DWORD    0x0
    ScreenColors    REG_DWORD    0x7
    TrimLeadingZeros    REG_DWORD    0x0
    WindowSize    REG_DWORD    0x190050
```

```
     LoadConIme     REG_DWORD     0x1
     PopupColors     REG_DWORD     0xf5
     QuickEdit     REG_DWORD     0x0
     WordDelimiters     REG_DWORD     0x0
     ColorTable10     REG_DWORD     0xff00
     ColorTable00     REG_DWORD     0x0
     ColorTable11     REG_DWORD     0xffff00
     ColorTable01     REG_DWORD     0x800000
     ColorTable12     REG_DWORD     0xff
```

Comparing Registry Keys

Comparing registry keys is done via the REG COPY command.

Syntax

```
REG COMPARE [ROOT\]RegKey [ROOT\]RegKey [/v ValueName] [Output] [/s]
REG COMPARE [ROOT\]RegKey [ROOT\]RegKey [/ve] [Output] [/s]
```

Wherein Output - /od (only differences) /os (only matches) /oa (all) /on (no output).

Example

```
@echo off
REG COMPARE HKEY_CURRENT_USER\Console HKEY_CURRENT_USER\Console\Test
```

The above program will compare all of the values between the registry keys HKEY_CURRENT_USER\Console & HKEY_CURRENT_USER\Console\Test.

Output

```
Result Compared:  Identical
The operation completed successfully.
```

If there is a difference between the values in either registry key, it will be shown in the output as shown in the following result. The following output shows that the value 'EnableColorSelection' is extra I the registry key 'HKEY_CURRENT_USER\Console'.

```
< Value: HKEY_CURRENT_USER\Console  EnableColorSelection REG_DWORD 0x0
Result Compared:  Different
The operation completed successfully.
```

20. Batch Script – Network

Batch Script has the facility to work with network settings. The NET command is used to update, fix, or view the network or network settings. This chapter looks at the different options available for the net command.

NET ACCOUNTS

View the current password & logon restrictions for the computer.

Syntax

```
NET ACCOUNT [/FORCELOGOFF:{minutes | NO}] [/MINPWLEN:length]
[/MAXPWAGE:{days | UNLIMITED}] [/MINPWAGE:days]
[/UNIQUEPW:number] [/DOMAIN]
```

Wherein

- **FORCELOGOFF** – Force the log-off of the current user within a defined time period.
- **MINPWLEN** – This is the minimum password length setting to provide for the user.
- **MAXPWAGE** - This is the maximum password age setting to provide for the user.
- **MINPWAGE** - This is the minimum password age setting to provide for the user.

Example

```
NET ACCOUNT
```

Output

```
Force user logoff how long after time expires?:       Never
Minimum password age (days):                          0
Maximum password age (days):                          42
Minimum password length:                              0
Length of password history maintained:                None
Lockout threshold:                                    Never
Lockout duration (minutes):                           30
Lockout observation window (minutes):                 30
Computer role:                                        SERVER
The command completed successfully.
```

NET CONFIG

Displays your current server or workgroup settings.

Syntax

```
NET CONFIG
```

Example

```
NET CONFIG
```

Output

```
The following running services can be controlled:
   Server
   Workstation
The command completed successfully.
```

NET COMPUTER

Adds or removes a computer attached to the windows domain controller.

Syntax

```
NET COMPUTER \\computername {/ADD | /DEL}
```

Example

```
NET COMPUTER \\dxbtest /ADD
```

Output

The above command will add the machine with the name dxbtest to the domain in which the windows domain controller exists.

NET USER

This command can be used for the following:

- View the details of a particular user account.
- Add a user account.
- Delete a user’s account.
- Modify a user’s account.

Syntax

```
Net user [username [password | *] [options]] [/DOMAIN]
username {password | *} /ADD [options] [/DOMAIN]
username [/DELETE] [/DOMAIN]
```

Example

```
NET USER
```

The above command shows all the accounts defined on a system. Following is the output of the above command.

```
User accounts for \\WIN-50GP30FGO75


-------------------------------------------------------------------------------
Administrator            atlbitbucket             Guest
The command completed successfully.
```

```
net user Guest
```

The above command shows the details of Guest account defined on a system. Following is the output of the above command.

```
User name                    Guest
Full Name
Comment                      Built-in account for guest access to the
computer/domain
User's comment
Country/region code          000 (System Default)
Account active               No
Account expires              Never

Password last set            1/4/2016 9:34:25 AM
Password expires             Never
Password changeable          1/4/2016 9:34:25 AM
Password required            No
User may change password     No

Workstations allowed         All
Logon script
User profile
```

```
Home directory
Last logon                   Never

Logon hours allowed          All

Local Group Memberships      *Guests
Global Group memberships     *None
The command completed successfully.
```

NET STOP/START

This command is used to stop and start a particular service.

Syntax

```
Net stop/start [servicename]
```

Example

```
NET STOP Spooler
```

The above command is used to stop the printer spooler service. Following is the output of the above command.

```
The Print Spooler service is stopping.
The Print Spooler service was stopped successfully.
NET START Spooler
```

The above command is used to start the printer spooler service. Following is the output of the above command.

```
The Print Spooler service is starting.
The Print Spooler service was started successfully.
```

NET STATISTICS

Display network statistics of the workstation or server.

Syntax

```
Net statistics [SERVER/WORKSTATION]
```

Example

```
Net statistics Server
```

Output

```
Server Statistics for \\WIN-50GP30FGO75



Statistics since 1/3/2016 9:16:28 PM



Sessions accepted                  0
Sessions timed-out                 0
Sessions errored-out               0

Kilobytes sent                     0
Kilobytes received                 0

Mean response time (msec)          0

System errors                      0
Permission violations              0
Password violations                0

Files accessed                     0
Communication devices accessed     0
Print jobs spooled                 0

Times buffers exhausted

  Big buffers                      0
  Request buffers                  0
```

NET USE

Connects or disconnects your computer from a shared resource or displays information about your connections.

Syntax

```
NET USE [devicename | *] [\\computername\sharename[\volume] [password | *]]
[/USER:[domainname\]username]
[/USER:[dotted domain name\]username]
[/USER:[username@dotted domain name]
[/SMARTCARD]
[/SAVECRED]
[[/DELETE] | [/PERSISTENT:{YES | NO}]]
```

where

- **\\computername\sharename** - This is the name of the share which needs to be connected to.

- **/USER** – This needs to be specified to ensure that the right credentials are specified when connecting to the network share.

Example

```
net use z: \\computer\test
```

The above command will connect to the share name \\computer\test and assign the Z: drive name to it.

21. Batch Script – Printing

Printing can also be controlled from within Batch Script via the NET PRINT command.

Syntax

```
PRINT [/D:device] [[drive:][path]filename[...]]
```

Where /D:device - Specifies a print device.

Example

```
print c:\example.txt /c /d:lpt1
```

The above command will print the example.txt file to the parallel port lpt1.

Command Line Printer Control

As of Windows 2000, many, but not all, printer settings can be configured from Windows's command line using PRINTUI.DLL and RUNDLL32.EXE

Syntax

```
RUNDLL32.EXE PRINTUI.DLL,PrintUIEntry [ options ] [ @commandfile ]
```

Where some of the options available are the following:

- /dl- Delete local printer.
- /dn - Delete network printer connection.
- /dd - Delete printer driver.
- /e - Display printing preferences.
- /f[file] - Either inf file or output file.
- /F[file] - Location of an INF file that the INF file specified with /f may depend on.
- /ia - Install printer driver using inf file.
- /id - Install printer driver using add printer driver wizard.
- /if - Install printer using inf file.
- /ii - Install printer using add printer wizard with an inf file.
- /il - Install printer using add printer wizard.
- /in - Add network printer connection.

- /ip - Install printer using network printer installation wizard.
- /k - Print test page to specified printer, cannot be combined with command when installing a printer.
- /l[path] - Printer driver source path.
- /m[model] - Printer driver model name.
- /n[name] - Printer name.
- /o - Display printer queue view.
- /p - Display printer properties.
- /Ss - Store printer settings into a file.
- /Sr - Restore printer settings from a file.
- /y - Set printer as the default.
- /Xg - Get printer settings.
- /Xs - Set printer settings.

Testing if a Printer Exists

There can be cases wherein you might be connected to a network printer instead of a local printer. In such cases, it is always beneficial to check if a printer exists in the first place before printing.

The existence of a printer can be evaluated with the help of the RUNDLL32.EXE PRINTUI.DLL which is used to control most of the printer settings.

Example

```
SET PrinterName=Test Printer
SET file=%TEMP%\Prt.txt
RUNDLL32.EXE PRINTUI.DLL,PrintUIEntry /Xg /n "%PrinterName%" /f "%file%" /q
IF EXIST "%file%" (
     ECHO %PrinterName% printer exists
) ELSE (
     ECHO %PrinterName% printer does NOT exists
)
```

The above command will do the following:

- It will first set the printer name and set a file name which will hold the settings of the printer.

- The RUNDLL32.EXE PRINTUI.DLL commands will be used to check if the printer actually exists by sending the configuration settings of the file to the file Prt.txt

22. Batch Script – Debugging

Very often than not you can run into problems when running batch files and most often than not you would need to debug your batch files in some way or the other to determine the issue with the batch file itself. Following are some of the techniques that can help in debugging Batch Script files.

Error Messages

To discover the source of the message, follow these steps:

Step 1: REM out the @ECHO OFF line, i.e. REM @ECHO OFF or :: @ECHO OFF.

Step 2: Run the batch file with the required command line parameters, redirecting all output to a log file for later comparison.

```
test.bat > batch.log  2>&1
```

Step 3: Search the file batch.log for the error messages.

Step 4: Check the previous line for any unexpected or invalid command, command line switch(es) or value(s); pay special attention to the values of any environment variables used in the command.

Step 5: Correct the error and repeat this process until all error messages have disappeared.

Complex Command Lines

Another common source of errors are incorrectly redirected commands, like for example "nested" FIND or FINDSTR commands with incorrect search strings, sometimes within a FOR /F loop.

To check the validity of these complex commands, follow these steps:

Step 1: Insert "command check lines" just before a line which uses the complex command set.

Following is an example wherein the ECHO command is inserted to mark where the output of the first TYPE command ends and the next one starts.

```
TYPE %Temp%.\apipaorg.reg
ECHO.================================================ TYPE %Temp%.\apipaorg.reg
| FIND
"[HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\TCPIP\Parameters\Interfa
ces\"
```

Step 2: Follow the procedure to find error message sources described above.

Step 3: Pay special attention to the output of the "simplified" command lines: Is the output of the expected format? Is the "token" value or position as expected?

Subroutines

Subroutines generating error messages pose an extra "challenge" in finding the cause of the error, as they may be called multiple times in the same batch file.

To help find out what causes the incorrect call to the subroutine, follow these steps:

Step 1: Add and reset a counter variable at the beginning of the script:

```
SET Counter=0
```

Step 2: Increment the counter each time the subroutine is called, by inserting the following line at the beginning of the subroutine:

```
SET /A Counter += 1
```

Step 3: Insert another line right after the counter increment, containing only the SET command; this will list all environment variables and their values.

Step 4: Follow the procedure to find error message sources described above.

Windows Versions

If you intend to distribute your batch files to other computers that may or may not run the same Windows version, you will need to test your batch files in as many Windows versions as possible.

The following example shows how to check for various operating system versions to check the relevant windows versions.

```
@ECHO OFF
:: Check for Windows NT 4 and later
IF NOT "%OS%"=="Windows_NT" GOTO DontRun
:: Check for Windows NT 4
VER | FIND "Windows NT" >NUL && GOTO DontRun
:: Check for Windows 2000
VER | FIND "Windows 2000" >NUL && GOTO DontRun
:: Place actual code here . . .
:: End of actual code . . .
```

```
EXIT
:DontRun
ECHO Sorry, this batch file was written for Windows XP and later versions only
```

23. Batch Script – Logging

Logging in is possible in Batch Script by using the redirection command.

Syntax

```
test.bat > testlog.txt 2> testerrors.txt
```

Example

Create a file called test.bat and enter the following command in the file.

```
net statistics /Server
```

The above command has an error because the option to the net statistics command is given in the wrong way.

Output

If the command with the above test.bat file is run as

```
test.bat > testlog.txt 2> testerrors.txt
```

And you open the file testerrors.txt, you will see the following error.

```
The option /SERVER is unknown.
```

The syntax of this command is:

```
NET STATISTICS
[WORKSTATION | SERVER]
```

More help is available by typing NET HELPMSG 3506.

If you open the file called testlog.txt, it will show you a log of what commands were executed.

```
C:\tp>net statistics /Server
```