

# Scripting guide for Windows administrators



## Updates and feedback

*This download was last updated on January 12, 2004. If you'd like to check for a more recent version, [click here](#). We also encourage you to [join our downloads discussion](#) and tell us what you think about this download, suggest improvements, and offer ideas for related tools you'd like us to create or that you'd like to submit for publication. Please include the name of this download in the subject line of your discussion post.*

Managing a Windows network can involve a lot of repetitive tasks, especially when that network includes large numbers of users and desktops. One of the best ways to automate these tasks is through the use of scripting. Windows includes two built-in methods of scripting: Windows shell scripting and VBS.

This compilation of articles by **Allen V. Rouse** introduces the role that scripts can play in Windows networks, shows how to get started with both Windows shell scripting and VBS, walks through a few advanced scripting techniques, and directs you to additional scripting resources.

## Table of contents

Understand the role of scripting in network administration .....	2
Getting started with Windows scripting languages and platforms.....	4
Windows shell scripting can expedite network admin tasks .....	7
First steps in VBS scripting for administrators .....	11
Improve administration by using the Shell and Network VBS objects .....	12
Improve efficiency of admin scripts with programming constructs .....	17
Beef up your admin scripts with these extra tools.....	21
These resources can help you write admin scripts .....	24
Listings.....	26

# Understand the role of scripting in network administration

Network administrators have used scripting since long before Windows or even DOS came on the scene. UNIX administrators, for instance, have been using shell scripting and its powerful capabilities for decades. Scripting can significantly ease the burden of network administration. But learning to create useful and effective scripts for networking tasks is not easy and requires a lot of patience and practice. Before you begin, it's important to have a good understanding of what scripting is and why it is so useful.

In the first article of this scripting guide, I will provide information to help you understand how scripting can play a role in network maintenance and management. In subsequent articles, I will talk about the various scripting languages and platforms, basic programming concepts for scripting, and using Windows shell scripting and Visual Basic scripting for network administration. I'll also point you toward other resources that are available for learning how to write scripts.

### What is scripting?

Simply stated, a script is a small, interpreted program that can carry out a series of tasks and make decisions based on specific conditions it finds. By "interpreted," we mean that when it is run, it is carried out one line at a time, as opposed to "compiled," which is the process of turning it into machine language before it is run. A script is created using ASCII text, so Windows Notepad or a similar text editor is the only tool required.

A number of scripting "languages" are available for you to choose from, each with its own capabilities and limitations. These languages include Windows native shell scripting, Visual Basic Scripting Edition, JavaScript, Kixtart, and Perl. Which one you choose will ultimately depend on a combination of the tasks required and your own experience and inclinations.

Each scripting language has a collection of commands or keywords and a set of rules on how to use them. The set of rules for writing a script in any given language is called the syntax. Once you learn the keywords and syntax, you can use a text editor to write the script and then save it with a file extension that is appropriate to the scripting language you are using. Some of the more common file extensions you will see are .bat, .cmd, .vbs, .js, and .kix.

### How is scripting used?

Scripting lets you automate various network administration tasks, such as those that are performed every day or even several times a day. For example, login scripts run every time a user logs in to the network and can perform tasks like mapping network drives for the user based on certain conditions, such as group membership. Another example of script use might be a situation where you want to have each Windows NT server create a new Emergency Restore Disk and then copy the contents of that disk to a network location.

Other tasks might need to be carried out only once, such as a modification to the registry, but to a large number of servers that are widely distributed geographically. In a case like that, you could create and distribute a single script to run the task on each server.

You can start scripts manually, but you can also start them automatically, either by a specific event or scheduled via the Windows Task Scheduler. Windows NT allows scripts to be run automatically each time a user logs in to the network. Windows 2000 goes much further and can be configured to automatically run separate scripts upon:

- Machine startup
- Machine shutdown
- User login
- User logout

# Windows scripting guide for administrators

You could, for instance, map specific network drives when a user logs in and then automatically copy that user's Favorites folder to a network share when he or she logs out so that the data is preserved in a central location.

## The scripting advantage

Scripting in network administration offers significant advantages. It allows you to:

- **Save time**—Scripts can carry out complex tasks and be invoked automatically, without the intervention of the network administrator, so the admin can concentrate on other tasks while the script runs.
- **Be consistent**—A script need be written only once and can then be invoked many times. It is much less error-prone than manually carrying out the task each time.
- **Be flexible**—Scripts can use decision-making logic to respond to different conditions. Rather than statically mapping a workstation to persistent drives, for example, network drives can be mapped in a variety of ways based on which user is logging on to the machine. You could write a script to check whether a file exists and delete it if it does, or display an error message if it doesn't. The only real limit to scripting is your imagination.

# Getting started with Windows scripting languages and platforms

You can think of a scripting platform as an environment in which a script can run. Given that a script is nothing more than a collection of text, there has to be some means for the computer on which the script is running to understand that text and carry out its instructions.

In Windows, there are two major scripting platforms to choose from: the Windows shell and the Windows Scripting Host (WSH). We'll cover the Windows shell first and then look at the WSH, along with its two scripting languages, VBS and JScript.

## Shell scripting

A shell is nothing more than an interface that allows a user to communicate with, or issue commands directly to, the operating system. The concept of a shell has been around in UNIX for many years. In fact, there are several shells in the UNIX world, each with its own features and commands that make it suitable for various tasks.

In Windows, there is no such diversity. You have only one shell, the Windows shell, which is built into the operating system. And you are undoubtedly already familiar with the interface, although you probably call it the command prompt or, if you're a real old-timer, perhaps the DOS prompt. Technically speaking, it's called a command shell and is run by executing the file `Cmd.exe`, found in `C:\Winnt\System32`. Probably the easiest way to run it is to simply click Start | Run, type `cmd` in the text box, and click OK, or create a shortcut to `Cmd.exe`.

The Windows shell comes with a set of built-in commands, many of which are well known and commonly used, such as `dir`, `copy`, `del`, `cd`, etc. Commands and their associated parameters are usually issued one at a time at the command line. More important for our purposes is the fact that commands can also be used in a batch mode. That is, using a text editor, you can write a separate command on each line, saving the finished product with the extension of either `.bat` or `.cmd`. This turns the text file into an executable that will be run as an interpreted program, carrying out each command one line at a time, in order. This is what we call shell scripting.

Although the Windows scripting language is far from being a full-scale programming language, it does come with some useful commands and features that allow it to have some of the flexibility you'd expect to find in a program. Some of these features are:

## Conditional processing

You can have your script test to see whether a certain condition exists, and if it does, do one thing, and if it doesn't, do something else.

## Error trapping

Every time a command is carried out, Windows generates an error level, with error level 0 being "no error." This allows you to include a provision in your script to gracefully exit from an error it might encounter.

## System variables

Information about a given computer and the user who is logged on to that computer can be found in the registry, at `HKEY_LOCAL_MACHINE` and `HKEY_CURRENT_USER`. Some of that information, which can be of use in scripting, is available in the form of system variables.

To get an idea of what is available, you can open the command shell and type the command `set`. This will display a list of all the system variables and their current values. These can then be referenced in a script by bracketing them with the percent symbol. For instance, `%username%` will refer to the username of whoever is currently logged on to the computer. An example of its use would be to copy the current user's Favorites folder and all subfolders on the local machine to that user's home folder on the server:

```
xcopy %userprofile%\favorites \\fileserver\home\%username% /s/y
```

# Windows scripting guide for administrators

I will talk more about these features in a future article devoted to shell scripting basics. You'll find a reference for all of the commands available for shell scripting in Windows Help. If you are using Windows 2000, click on Start | Help. In the Search tab, type *command reference* and click List Topics. Then, under Select Topic To Display, double-click on the topic Windows 2000 Command Reference Main Page.

In Windows NT, click on Start | Help. In the Find Tab, type *command*. Then, in the Pick A Topic pane, select Commands Index and click the Display button.

## Windows Scripting Host

The Windows Scripting Host (WSH) is a set of three files (Wscript.exe, Cscript.exe, and Wsh.ocx) that provide an environment for other scripting languages to run in. Built into the WSH are two "engines" for the scripting languages Visual Basic Scripting Edition (VBS) and JScript, which is a Microsoft version of JavaScript. You can also load other engines for such scripting languages such as Perl or REXX, if you want.

Although the shell scripting language remains a fixed part of the operating system, WSH can be separately updated and upgraded, since it exists as separate files. In addition, it can be installed on several versions of Windows. To download the latest version of WSH, go to the Microsoft Windows [Script Page](#) and follow the link to the download page. To determine which version is currently installed, type *cscript* at the command shell.

The WSH makes use of a rather strange concept called an object model, which can take some getting used to for a newcomer to scripting and programming. You can think of an object as a tool that you use to accomplish certain tasks. Each object has a set of methods associated with it. You can think of methods as the functions or capabilities of each tool, or object. The root object for WSH is called WScript, and from it, other objects can be created and used within scripts to accomplish tasks.

Both VBS and JScript are object-based languages, and each uses its own object model that works in conjunction with the WSH object model.

## Visual Basic Scripting Edition

In future articles, I will discuss the basics of using VBS for writing scripts. For the moment, let's take a quick look at what it can do.

VBS is a subset of the full-blown programming language Visual Basic. (Another subset of Visual Basic, called Visual Basic for Applications—VBA—is used primarily to create macros for Microsoft Office applications.) Using the objects and methods found in both the Windows Scripting Host and in VBS, along with correct VBS syntax, you can create a VBS script.

You do this by using a text editor such as Notepad to write out a set of commands and then save the finished script with the file extension .vbs. Once this file is created, you can execute it just as you would any other executable file. For example, you can:

- Double-click on the icon.
- Type the filename at the command prompt.
- Type the name of the file on its own line within a shell script.

## JScript

JScript is the Microsoft implementation of JavaScript. JScript and JavaScript are not subsets of the Java programming language. However, like VBS, JScript is an object-based scripting language that can be used to automate tasks for network administration.

You create a JScript file in the same way you create a VBS file, except that you must save it with the file extension .js. The default icon for a JScript file looks just like the icon for a VBS file, except that the VBS icon is blue and the JScript icon is yellow. You execute a JScript file in the same way you execute a VBS file.

# Windows scripting guide for administrators

## Which one to use?

The choice of whether to use VBS or JScript is largely a matter of preference. If you've worked with JavaScript on Web sites, you may be more comfortable using JScript for network administration scripting. On the other hand, if you have some experience with Visual Basic, VBA, or VBS Web scripting, you will probably prefer using VBS for networking scripts.

It is even possible to use both scripting languages in the same script, if you save it with the file extension .wsf (Windows Script File). However, learning both languages requires twice the time, so it's probably best to stick to just one. For the purposes of this guide, I will concentrate on VBS rather than JScript simply because that is my preference.

# Windows shell scripting can expedite network admin tasks

Some of the more common Windows scripting languages and platforms include Windows shell scripting, Visual Basic Scripting (VBS), and JScript, as I discussed previously. Now we're going to take an in-depth look at Windows shell scripting as it relates to network administration. For the purposes of this discussion, I'm going to assume that you understand the basics of using the Windows command line, including the use of parameters and switches, such as the following:

```
del? myfile.txt /f
```

Here, "del" is the command, "myfile.txt" is the parameter (which provides required information to the del command—in this case, which file to delete), and "/f" is the switch, which modifies the behavior of the command so that it forces the deleting of read-only files.

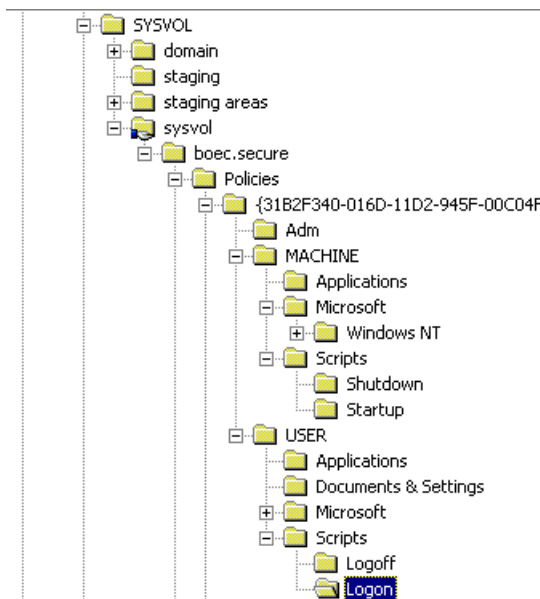
## Logon scripts

Probably the most common use for shell scripting is logon scripts. A logon script is used to configure the Windows environment for a user at the time of logon and is usually specific to a group of users. For instance, members of the Finance group may automatically map network drives to the finance network share folder, while the Marketing group may automatically map their network drives to the marketing network share.

To make this work, a script is created for each user or group of users and then copied to the appropriate server location (based on the version of Windows). In Windows NT, the script file is normally placed in C:\Winnt\System 32\Repl\Export\Scripts (or Import, depending on how you have configured replication). You then point to that file in the User Account Properties dialog box.

When using Active Directory, you deploy the logon script via a Group Policy. First, copy the script file to the Sysvol subfolder. **Figure A** shows where to access this subfolder.

FIGURE A

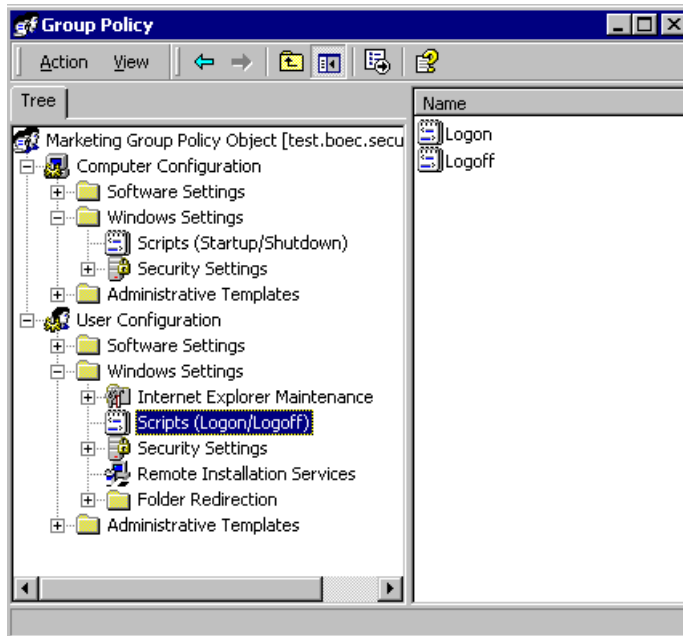


## Finding the Scripts folders in Sysvol

Note that you also have the choice of scripts for logoff, startup, and shutdown. You can then directly edit the Group Policy object for a given container to include that script, as shown in **Figure B**.

# Windows scripting guide for administrators

FIGURE B



Edit the script in a specific container in Active Directory.

## Commands in logon scripts

One of the most useful commands to use in a logon script is NET USE. It's just one subset of many available NET commands.

The NET USE command allows you to establish drive mappings. When used in a logon script, it can tailor the drive mapping to a specific user or group of users. For instance, let's say that the marketing department requires a drive mapping to the marketing folder on Server3, another drive mapping to the admin folder on Server2, and another drive mapping to the individual user's home folder found in the "home folder" share on Server1. Here is an example of how you would script that:

```
NET USE F: \\server3\marketing PERSIST:NO
NET USE G: \\server2\admin PERSIST:NO
NET USE H: "\\server1\home folder\%username%" PERSIST:NO
```

The keyword PERSIST at the end of each line addresses whether to reconnect the drive mapping at the next startup. Most often, you do not want this to happen because another user may require different drive mappings.

You also should note a couple of things in the third line of code. First, there are quotation marks around the path. This is required because there is a space within the pathname. Second, I used the environmental variable %username%. When a user logs on, that username is stored temporarily in the registry and is available within the Windows shell. (To verify this, type *echo %username%* at the command line and you should see your own logon name provided as output.) If each user has a folder that is named the same as that person's logon name, you can use that to automatically map a drive to the home folder.

The %username% variable is one of several environment variables that are created automatically by Windows and that can be used similarly in your scripts. To see a complete list, you can type *set* at the command prompt. You can also create your own variables with this command. To find out how, type *set /?* at the command prompt.



# Windows scripting guide for administrators

## WINDOWS COMMANDS

To see a complete list of the commands and information on how to use them, refer to Windows Help. If you are using Windows 2000, click on Start | Help. In the Search tab, type *command reference*. Under Select Topic To Display, double-click on Command Reference Main Page. In Windows NT, click on Start | Help and type *command* in the Find tab. Then, under Pick A Topic, select Commands Index and click the Display button.

Another useful logon script command is NET TIME, which is used to synchronize time on a network to a time server. You can use a number of switches with this command. Again, you can check the command reference in Windows Help. The simplest use of NET TIME might look something like this:

```
NET TIME \\timesvr8 /SET /YES
```

This tells the computer to synchronize its time with that on the server named timesvr8. The /YES switch tells it to force the synchronization even if the named server is not a time source server.

In a future article, I will discuss certain programming constructs you will find useful in scripting, including conditional processing. Here's a look at one form of conditional processing.

Administrators in Novell NetWare have long appreciated the ability to tell the logon script to do something if the user is a member of a certain group by using the statement IF MEMBER OF. Although no such statement is available in Windows scripting, there are two possible workarounds, depending on whether you are administering a Windows NT domain or a Windows 2000 domain with Active Directory.

The Windows NT Resource Kit includes a utility named IFMEMBER.EXE, which can be used for the same purpose as the NetWare IF MEMBER OF. Unfortunately, its use is rather convoluted, requiring yet another construct called ERRORLEVEL. Here's how it works. Lets say that if the user logging on is a member of the both the Marketing group and the Managers group, you want to map a drive to a share that only marketing managers have access to. First, you would include the line:

```
IFMEMBER marketing managers
```

The IFMEMBER utility, assuming that it is in a search path, will check for membership in all groups listed and then exit and store the number of groups the user is a member of in a special variable called ERRORLEVEL. In this case, that number should be 2. So the entire process would look like this:

```
IFMEMBER marketing managers
IF NOT ERRORLEVEL 2 EXIT
NET USE J: \\SERVER4\MKTGMGR? /PERSIST: NO?
```

If the user is a member of both groups, the value stored in ERRORLEVEL will be 2. If the value is not 2, the script will end. But if the value is 2, it will execute the next statement.

The IFMEMBER.EXE utility may also be used in a Windows 2000 domain, but if you are using Active Directory, you have another option. Rather than reference groups in logon scripts, you can design your Organizational Units along the same lines as those groups. That way, you can create Group Policy Objects linked to each OU, with a specific logon script for that OU, without having to use the IFMEMBER utility.

## Using shell scripting for other purposes

As I mentioned, a logon script is designed to be invoked every time a user logs on to the network. There are times, however, when you will find that a script can be useful even if it is used only once. For example, suppose that your organization has just completed work on a Geographic Information System (GIS) that displays maps. Any GIS involves a large number of "shape" files and images, often in a complex directory structure. They will run best if installed locally on the workstation, so you decide that you want to copy these files, with directory structures intact, from a network share to the local hard drive of each workstation.

You could, of course, sit down at each computer, open the command prompt, and manually type each line one at a time. But obviously, it would greatly simplify things to write one script that contains every command that must be issued, each on its own line, and simply run that script at each workstation.

# Windows scripting guide for administrators

Using the COPY command or the more versatile XCOPY command, you could create a script that would accomplish the job, copy it to a removable medium such as a floppy disk, and run the script at each station by typing a single command. The advantages are twofold:

- It will save you time, since you don't have to type every command at every workstation.
- It will help prevent the inevitable errors caused by mistyping a command or a pathname.

## Best practices

When you write Windows shell scripts, you should keep these best practices in mind:

- Always test your scripts before you use them in a production environment.
- Always document your scripts, even if they appear very simple. When a script encounters the keyword REM (for "remark"), it will ignore that entire line. You can therefore use that keyword to add remarks to your script for documentation. At the very least, you should include the purpose of the script, with the date and the name of the person who created it. Remember that what may seem obvious to you at the time may not be obvious to someone else. For that matter, it may not even be obvious to you a year from now.
- Remember that each command line in your script will be displayed onscreen when it is run, unless you turn that feature off with the command @ECHO OFF.

## Moving on

To really become proficient at creating Windows shell scripts, you need to understand the available commands and their syntax. Study the command reference in Windows Help. Take a look at each command and see which ones might help you accomplish the tasks you need done. Each command has a Related Topics link. Most of those include two additional useful links: one that provides examples of that command in use and one that has further notes on the command.

In addition, you should practice and try out various commands and options as much as possible until you become proficient. Don't be afraid to make mistakes (provided you are practicing in a test environment). That's the best way to learn about the variables involved in this process.

# First steps in VBS scripting for administrators

Even the most experienced network administrator may find the powerful VBS scripting somewhat intimidating. VBS scripting is a subset of Microsoft Visual Basic called Visual Basic Scripting Edition. It looks quite different from shell scripting and, in fact, uses a very different concept—one that's based on objects and methods.

But network administrators don't have to be master programmers to write effective and useful VBS scripts. It's just a matter of taking a few steps into the unknown. I'm going to introduce you to VBS scripting by discussing the concept of using objects and methods to accomplish a task. We will then apply this concept to work with the Windows file system to carry out a simple file management task.

## Objects and methods

Probably the main reason why VBS code appears so strange to someone who is used to writing batch files is that it makes use of a syntax that combines objects and methods to accomplish its tasks. All of the common programming concepts are there—variables and constants, error trapping, conditional processing, and so on. But to get anywhere with VBS, you must start with objects.

The platform for VBS scripting is the Windows Script Host (Wscript.exe). It comes with a set of tools, each of which has specific capabilities. In VBS, a tool is called an object, and the various capabilities of a tool are called methods. So to accomplish a task, you combine an object and a method. The way this is done in VBS syntax is in a dot format like this:

```
object.method
```

The root object in the Windows Script Host (WSH) is called Wscript. There are 13 other WSH objects, but we don't need to go into those now. The main thing here is to understand the concept of using objects and methods to carry out a task. The Microsoft MSDN Web site provides more information on the [WSH Object Model](#).

## Introducing FileSystemObject

The easiest way to understand the use of objects and methods is to become familiar with FileSystemObject, the object used to interact with the Windows file system. However, because it is not one of the core WSH objects, it can't be used directly. It is actually invoked by WSH from a file called the Scripting Runtime Library (%windir%\system32\scrn.dll).

Once invoked, FileSystemObject can be used to create, copy, or delete files or folders. You invoke it through a process called "instantiating an object," which is basically creating the object and then assigning it to a variable.

We will use the Windows Script Host root object Wscript to create the FileSystemObject object (yes, that does sound redundant) and simultaneously assign it to a variable. Here's how.

One of the methods in the Wscript object is called CreateObject. So using our dot notation, we can write Wscript.CreateObject to accomplish our task. We just need to tell it that we want to create the FileSystemObject found in the Scripting Runtime Library:

```
Wscript.CreateObject("Scripting.FileSystemObject")
```

However, that's only half the job of instantiating FileSystemObject; we also have to simultaneously assign it to a variable. In programming, a variable is simply a name you give to something so that you (and the program) can keep track of it more easily. Since we are creating the FileSystemObject, we might use FSO as the variable name. But there are many kinds of items besides objects that can be given names. To identify this as the name of an object, we'll adopt Microsoft's recommended naming convention and use the variable name objFSO.

We are now close to the point where we can actually do something. But in Visual Basic, you can't use a variable name until you declare it and assign something to it. To declare a variable, we use a separate statement with the keyword Dim, like this:

```
Dim objFSO
```

# Windows scripting guide for administrators

Then, to assign something to the variable, we use the keyword `Set` and an equal sign. So the entire process of instantiating the `FileSystemObject` will look like this:

```
Dim objFSO
Set objFSO = Wscript.CreateObject("Scripting.FileSystemObject")
```

## Using FileSystemObject

Now that we have instantiated `FileSystemObject` and assigned it to a variable called `objFSO`, what can we do with it? Remember that objects come with a set of capabilities called methods. We can use those methods to manipulate the file system if we know the exact syntax. Some of the useful methods for `FileSystemObject` are listed in **Table A**.

TABLE A

CopyFile	Copies files to another location
DeleteFile	Deletes files
FileExists	Returns <i>True</i> or <i>False</i> depending on whether the file exists
MoveFile	Moves files to another location
CopyFolder	Copies folders to another location
DeleteFolder	Deletes folders
FolderExists	Returns <i>True</i> or <i>False</i> depending on whether the folder exists
MoveFolder	Moves folders to another location

Let's say that we want to write a script that will copy a file named `MyFile.txt` from a local drive to a network share. Good programming technique dictates that we first test to make sure that the file exists before we execute the `Copy` command. [Listing A](#) shows how we would do that.

Once we instantiated the `FileSystemObject` as `objFSO`, we were able to invoke two of its methods (`FileExists` and `CopyFile`) to accomplish a task. We also invoked a `Wscript` method (`Echo`) to display a message in case of an error. In addition, note that we invoked those methods using the variable name `objFSO` rather than the object name `FileSystemObject`.

Of course, the exact syntax is important, such as the `If-Then-Else` construct and the use of parentheses and quotation marks. But the point here is to understand the use of objects and methods to accomplish a task.

## Releasing FileSystemObject

Good programming technique requires you to do one more thing before leaving the VBS script. After using objects you have instantiated, you should always release them before ending the script. You do that by assigning *nothing* to the variable name you created—literally. Here's what it looks like:

```
Set objFSO = Nothing
```

So now, if we put the whole thing together and include, as we always should, documenting comments, it would look something like [Listing B](#).

## Just the beginning

As most network administrators know, you could accomplish this same task with a regular batch file or shell script. If that were all VBS scripting could do, there wouldn't be much point in taking the trouble to learn it. But VBS scripting can ultimately do a great deal more than shell scripting, including reading and writing to files, reading and writing to the Registry, creating shortcuts, and assigning icons. Once you understand the concept of using objects and methods, you will have taken the first step toward mastering this powerful tool.

# Improve administration by using the Shell and Network VBS objects

# Windows scripting guide for administrators

When you start working with the Windows Scripting Host and VBS scripting, you must understand the concept of objects and methods. Previously, I discussed how to instantiate, or create, an instance of the FileSystemObject object and use it to manipulate the Windows file system. Now, I'm going to discuss two other objects that are important in network admin scripting, WshShell and WshNetwork. We'll use these two objects to create shortcuts, write to the registry, and map network drives.

## Object recap

Objects, which can be thought of as tools, have what we call *methods*—certain things the tools can do. In addition, objects have *properties*. Just as the name implies, properties are characteristics of objects and, like methods, can be used to good advantage in VBS.

## Use the WshShell object to create a shortcut

To create the Shell object, we use the CreateObject method of WScript:

```
Dim objShell
Set objShell = WScript.CreateObject("WScript.Shell")
```

Once the Shell object has been created, it includes the properties and methods shown in **Table A**.

TABLE A

Properties	Methods
CurrentDirectory	AppActivate
Environment	CreateShortcut
SpecialFolders	Exec
?	ExpandEnvironmentStrings
?	LogEvent
?	Popup
?	RegDelete
?	RegRead
?	RegWrite
?	Run
?	SendKeys

## WshShell object

Using the Shell object and some of these properties and methods, we are going to create a VBS script that will create a shortcut to Notepad on the user's desktop. First, notice that one of the properties is called SpecialFolders. This property requires an argument telling it which special folder to use, such as MyDocuments, Favorites, or StartMenu. For our script, we'll use Desktop.

Second, notice that one of the methods for WshShell is CreateShortcut. If you use that method, you end up with an object called WshShortcut. Its properties are shown in **Table B**.

# Windows scripting guide for administrators

TABLE B

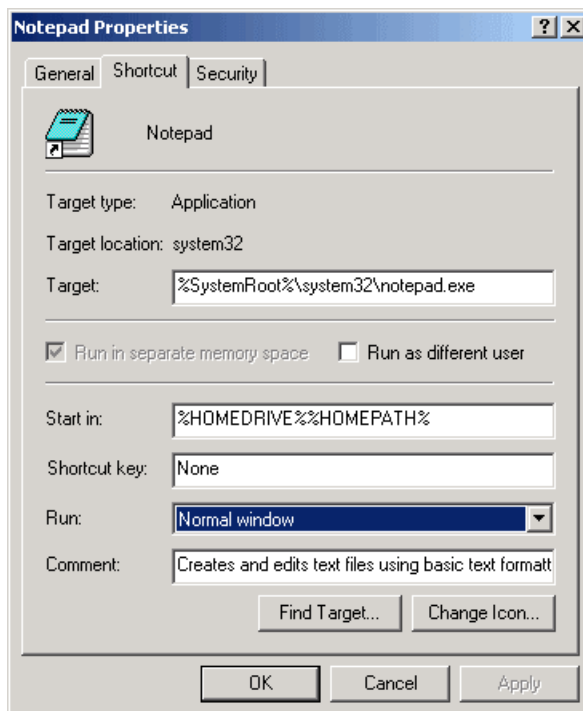
## Properties

Arguments  
Description  
Hotkey  
IconLocation  
TargetPath  
WindowStyle  
WorkingDirectory

## WshShortcut object

Where do these properties come from? If you right-click on a shortcut and select Properties from the context menu, you'll see something similar to **Figure A**.

FIGURE A



Items in the Properties dialog box correspond to WshShortcut properties. For instance, the Target text box corresponds to the TargetPath property in the WshShortcut object. The Run drop-down list lets you select either Normal Window, Run Minimized, or Run Maximized. These options are part of the WindowStyle property in WshShortcut. The only property we are required to specify is TargetPath.

Now let's look at how we can create a shortcut with VBS. First, we have to remember that shortcuts to files or folders (as opposed to URLs) have the extension .lnk. We will need to specify that we want to link to C:\WINNT\notepad.exe and store that link (shortcut) in the user's Desktop folder. Here is what we will do:

1. Instantiate the WshShell object.
2. Use WshShell to create a shortcut.
3. Specify where the link is located (using the TargetPath property of WshShortcut).

# Windows scripting guide for administrators

## 4. Save the shortcut.

[Listing A](#) shows how this script will look.

Note the various uses of parentheses and quotation marks in this script. Note also that in creating the shortcut, we specified the target folder using its variable name and concatenated the link name using an ampersand (&). In specifying the target path, we used the environment variable %windir%, which is normally C:\WINNT. The entire path must be enclosed in quotes.

## Use the WshShell object to work with the registry

As we saw in Table A, the Shell object contains the following methods:

- RegWrite
- RegRead
- RegDelete

We can use these methods to manipulate the registry of a machine with a VBS script—that is, read, write, and delete registry keys and values. The syntax for this is quite simple. Let's say that in the HKEY\_CURRENT\_USER hive (HKCU), we want to create a key with the name "AnewKey" and assign to it the value of "NewValue." We would first instantiate the Shell object and then use the RegWrite method as follows:

```
Dim objShell
Set objShell = WScript.CreateObject("WScript.Shell")
ObjShell.RegWrite "HKCU\AnewKey", "NewValue"
```

The same key and value can be removed using the RegDelete method.

## Use the WshNetwork object to map a network drive

**Table C** shows the properties and method of the WshNetwork object. We'll use the MapNetworkDrive method in this example.

[TABLE C](#)

Properties	Methods
ComputerName	AddWindowsPrinterConnection
UserDomain	AddPrinterConnection
UserName	EnumNetworkDrives
?	EnumPrinterConnections
?	MapNetworkDrive
?	RemoveNetworkDrive
?	RemovePrinterConnection
?	SetDefaultPrinter

## WshNetwork object

Let's say that we want to assign the network share \\Server1\Admin to drive G:. We first instantiate the network object and then use the MapNetworkDrive method as follows:

```
Dim objNetwork
Set objNetwork = WScript.CreateObject("WScript.Network")
ObjNetwork.MapNetworkDrive "G:", "\\Server1\Admin"
```

## The sky's the limit

Using the FileSystemObject, Shell, and Network objects and their associated properties and methods, you can automate a great many tasks to ease the burden of routine work.

## Windows scripting guide for administrators

To learn more about VBS and how you can use the objects, methods, and properties to your advantage, check out Microsoft's site for specifics such as the [VBS User's Guide and Language Reference](#) and references on the individual objects [Shell](#), [Network](#), and [FileSystemObject](#).



# Improve efficiency of admin scripts with programming constructs

With Windows shell scripting and VBS, you can map network drives, copy files or folders, edit the registry, create shortcuts, and perform a host of other useful network administrative tasks. Earlier, I alluded to some programming constructs that let you extend the functionality of your scripts. Now it's time to take a look at these programming constructs, see how they are used, and examine their syntax in both shell scripting and VBS. The constructs I'll discuss are conditional processing, error trapping, and iterative processing.

### Conditional processing in Windows shell scripts

Conditional processing is the ability to have your script test whether a certain condition exists or is true, such as determining whether a variable is equal to a string or a specific file is present. Once the script determines whether the condition is true or false, it can carry out one command if the condition does exist and carry out another command if it does not exist.

In shell scripting, the conditional processing construct consists primarily of these elements:

- IF statement
- GOTO statement
- Labels

The IF statement is used to test a condition. This statement may include a command to be carried out if the condition is true, such as:

```
IF %SYSTEMROOT% == "C:\WINNT" COPY new.dll C:\WINNT\SYSTEM32
```

You can also include a command to be carried out if the condition is false. To do this, you add an ELSE clause to the IF statement:

```
IF %USERNAME% == "Johnny" echo "Hello Johnny" ELSE echo "Who are you?"
```

In addition, you have the option of branching to another part of the script to carry out a command. This is especially useful in more complex processing. To branch, you must include a label, which is basically a named place marker in your script. For example, if your script contains commands that can't be carried out on a Windows 9x operating system, you might want to test to see that the operating system is Windows NT or higher first. If it is not, you do not want the script to run at all. To accomplish this, you could use a label called EXIT and the GOTO statement:

```
IF NOT %OS% == "WINDOWS_NT" GOTO EXIT
[Body of script here]
GOTO EOF
:EXIT
ECHO "You must be using Windows NT or higher."
```

A little explanation is called for here. First, note that a colon must precede the label (EXIT) in the script and that I used two equal signs (==) instead of one (=). The double equal sign is called a comparison operator and must be used with IF statements. I used a GOTO statement to tell the script to branch to the label. Remember that the script will carry out each line in order until it reaches the end, unless it is told otherwise. This means that if the condition is true, it will still carry out that last ECHO statement in the script. Of course, I don't want to echo that message if the condition is true, so I used the GOTO EOF (End of File) statement before the EXIT label. This tells the script to end there and not carry out any more commands—in this case: *ECHO "You must be using Windows NT or higher."*

You can always use NOT to invert any test for a condition. Thus, in addition to the basic IF statement, you can use the following:

```
IF (NOT) DEFINED [Tests to determine whether a given variable exists]
IF (NOT) EXIST [Tests to determine whether a given file or folder exists]
```

The Command Reference in Windows Help provides details and notes on using these statements.

# Windows scripting guide for administrators

## Dealing with errors in Windows shell scripts

Every time a Windows shell command completes its task, it exits with a specific code. Commands that are completed with no errors exit with the code of 0, while any other exit code indicates that an error occurred. For instance, say that you execute the following command:

```
DEL myfile.txt
```

If the file was successfully deleted, the exit code will be 0. But if that file doesn't exist, or if it has a read-only attribute, the command can't delete it and the error code will be something other than 0. A variable called `ERRORLEVEL` lets you access that code, and you can use that in conjunction with the `IF` statement to trap errors. Here's an example:

```
DEL myfile.txt
IF NOT ERRORLEVEL 0 GOTO EXIT
ECHO "File deleted."
GOTO EOF
:EXIT
ECHO "Could not delete file."
```

## Conditional processing in VBS

In VBS, the syntax for conditional processing is a little different. To execute a straightforward `IF` statement, you type something like this:

```
IF condition THEN
    one or more statements
END IF
```

Notice that you must include the keyword `THEN` and explicitly end the `IF` statement. In addition, VBS will allow you to include statements that will be executed if the condition is false, as well as statements to execute if the condition is true. This is done with the `ELSE` clause, as shown here:

```
IF count = 10 THEN
    Wscript.echo "Count is ten."
ELSE
    Wscript.echo "Count is not ten."
END IF
```

VBS also has a slightly more efficient method of performing conditional processing, known as the `SELECT CASE` statement. It allows the script to evaluate a condition, as you might guess, on a case-by-case basis. Here's an example:

```
SELECT CASE count
    CASE count = 1
        Wscript.echo "Count is one."
    CASE count = 2
        Wscript.echo "Count is two."
    CASE count = 3
        Wscript.echo "Count is three."
END SELECT
```

## Iterative processing in shell scripting

Iterative processing, also called looping, means repeating one or more commands until some goal is reached. In shell scripting, the syntax is not as obvious as other constructs and is therefore a little harder to learn. For iterative processing you use the `FOR DO` statement, and the syntax looks like this:

```
FOR /switch %%variable in (set) DO command
```

The `/switch` element can be one of four switches:

- `/l` loops through a range of values.
- `/f` loops through or parses a string.
- `/d` loops through the files of a folder.
- `/r` loops through all subfolders.

The `%%variable` is any letter of the alphabet, such as `%%a` or `%%z`. The `FOR` command will increment the value of this variable each time it passes through the loop.

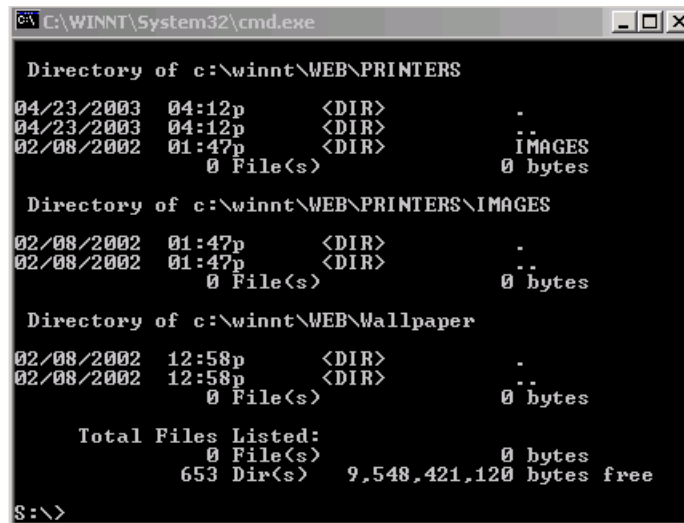
# Windows scripting guide for administrators

Look at how you might use this construct for a network admin task. Suppose that you need to print a list of all the subfolders in the C:\WINNT folder. You could do that using a straight shell command such as:

```
DIR C:\WINNT /AD /S
```

The output would look similar to **Figure A**.

FIGURE A



```
C:\WINNT\System32\cmd.exe

Directory of c:\winnt\WEB\PRINTERS
04/23/2003  04:12p    <DIR>        .
04/23/2003  04:12p    <DIR>        ..
02/08/2002  01:47p    <DIR>        IMAGES
               0 File(s)                0 bytes

Directory of c:\winnt\WEB\PRINTERS\IMAGES
02/08/2002  01:47p    <DIR>        .
02/08/2002  01:47p    <DIR>        ..
               0 File(s)                0 bytes

Directory of c:\winnt\WEB\Wallpaper
02/08/2002  12:58p    <DIR>        .
02/08/2002  12:58p    <DIR>        ..
               0 File(s)                0 bytes

Total Files Listed:
               0 File(s)                0 bytes
               653 Dir(s)    9,548,421,120 bytes free

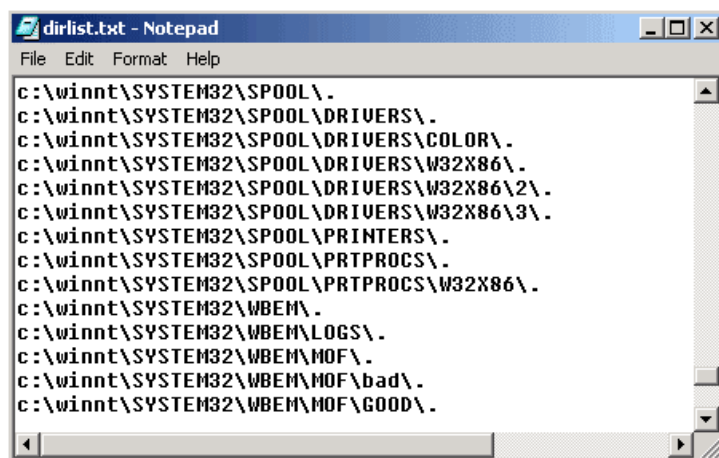
$:>
```

However, this may not be the most useful format. So I create a script with the following commands:

```
@echo off
FOR /r c:\winnt %a in (.) do echo %a > dirlist.txt
```

The resulting output, redirected to a text file, will appear similar in format to **Figure B**.

FIGURE B



```
dirlist.txt - Notepad
File Edit Format Help

c:\winnt\SYSTEM32\SPool\
c:\winnt\SYSTEM32\SPool\DRIVERS\
c:\winnt\SYSTEM32\SPool\DRIVERS\COLOR\
c:\winnt\SYSTEM32\SPool\DRIVERS\W32X86\
c:\winnt\SYSTEM32\SPool\DRIVERS\W32X86\2\
c:\winnt\SYSTEM32\SPool\DRIVERS\W32X86\3\
c:\winnt\SYSTEM32\SPool\PRINTERS\
c:\winnt\SYSTEM32\SPool\PRTPROCS\
c:\winnt\SYSTEM32\SPool\PRTPROCS\W32X86\
c:\winnt\SYSTEM32\WBEM\
c:\winnt\SYSTEM32\WBEM\LOGS\
c:\winnt\SYSTEM32\WBEM\MOF\
c:\winnt\SYSTEM32\WBEM\MOF\bad\
c:\winnt\SYSTEM32\WBEM\MOF\GOOD\
```

## Iterative processing in VBS

Iterative processing in VBS is frankly more useful and, as an added bonus, the syntax is actually easier. Four forms of looping are available in VBS:

- DO WHILE will create a loop that continues to process as long as a condition is true.
- DO UNTIL will create a loop that continues to process until a condition becomes true.
- FOR NEXT will create a loop that executes a specific number of times.
- WHILE WEND will create a loop that continues to process as long as a condition is true.

# Windows scripting guide for administrators

The WHILE...WEND loop is similar to DO...WHILE, but it is not used much, due to the general popularity of DO...WHILE. The basic formats for the other three constructs are:

```
DO WHILE condition
    Repeating commands
LOOP
-----
DO UNTIL condition
    Repeating commands
LOOP
-----
FOR a = 1 TO 10
    Repeating commands
NEXT
-----
```

It's important to understand that in both the DO...WHILE loop and the DO...UNTIL loop, the condition must change at some point. Otherwise the loop will never end. The FOR...NEXT loop shown above will repeat 10 times.

## Be constructive

Using the programming constructs I have discussed here, you can greatly improve the efficiency and functionality of your scripts. You can have your scripts make decisions without your intervention, recognize errors and take action based on them, and quickly perform repetitive tasks. As with all elements of scripting, developing the skills to use these constructs will take practice, but it will be well worth the trouble.

# Beef up your admin scripts with these extra tools

Earlier I mentioned IFMEMBER.EXE, a valuable Windows Resource Kit tool you can use to refine login scripts. Microsoft also provides a number of other tools that will help you improve your administration scripts. I'm going to show you where to find these tools and introduce a few that you may find useful. I'll also tell you about a particularly handy tool called [KiXtart](#), which you can use to enhance your scripts.

### Running executables in scripts

The utilities that Microsoft provides are command line executable files that come with their own syntax. To use them, you must first know how to run an executable file within a script.

Executing a utility within Windows shell scripts is fairly straightforward. You know that you can execute a command line program at the command prompt. Since shell scripts are basically a series of command line instructions, you can simply include the executable on its own line in the script. If it's not in the default path, just include the path as part of the command line.

For instance, one of the utilities that Microsoft provides is INUSE.EXE, a tool that can replace files that are currently in use by the operating system. To run it, you must specify the name of the source file and the name of the file to be replaced. For example, you might include a line like this in your script:

```
"C:\Program Files\Resource Kit\inuse.exe" f:\test1.dll \\SERVERA\Winnt\System32\test1.dll
```

It's not quite that easy in Visual Basic Scripting, but it's not too difficult either. First, to run a command line utility with a VBS script, you must access the *Run* method within the WshShell Object. The syntax for that is:

```
Wscript.Run (command, [WindowStyle], [WaitonReturn])
```

In this case, *command* is the complete command line utility with parameters, *WindowStyle* is an optional parameter that controls the style of the command window, and *WaitonReturn* is either true or false, specifying whether the script should wait until the utility completes processing before continuing with the script. Running INUSE.EXE within a VBS script might look something like this:

```
Set ObjWS = WScript.CreateObject("WScript.Shell")
ObjWS.Run ("INUSE.EXE f:\test1.dll \\SERVERA\Winnt\System32\test1.dll", 0,"true")
```

### Head for the tool shed

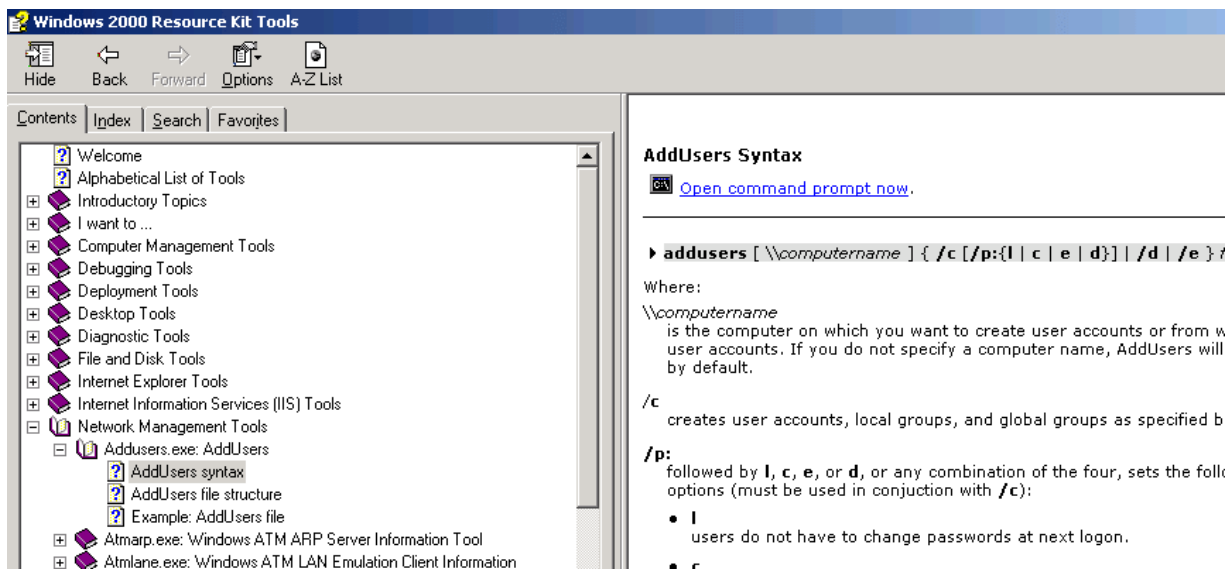
So, now that you know how to use these utilities within a script, you need to know where to go to get them. The best place to go is the Windows Resource Kit. Both the [Windows NT Resource Kit](#) and the [Windows 2000 Resource Kit](#) have the tools. You can also download many of the individual tools [here](#).

The Resource Kits generally provide you with not only the executable file, but also with complete information on how to use the tools, along with examples.

**Figure A**, for instance, shows the syntax for ADDUSERS.EXE, a command line utility you can use to create users and groups within a script.

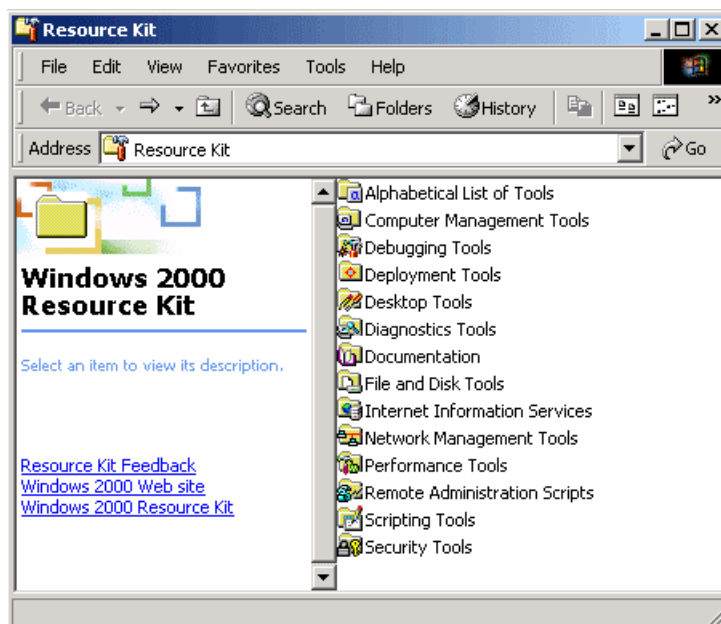
# Windows scripting guide for administrators

FIGURE A



The Resource Kits conveniently group the tools by functionality, as shown in **Figure B**. You can use the Search tab if you need to find a specific tool.

FIGURE B



**Table A** lists a few of the many Resource Kit tools you might find useful in your scripts for network administration.

TABLE A

<b>Delpref.exe</b>	Deletes user profiles from a Windows 2000 computer
<b>Delsrv.exe</b>	Unregisters a service
<b>Regini.exe</b>	Edits the registry with a script
<b>Logoff.exe</b>	Automatically logs off when used in an unattended installation

# Windows scripting guide for administrators

**Permcopyp.exe** Copies share and file permissions from one share to another

**Now.exe** Stamps current date and time

## Also consider KiXtart

The Windows Resource Kits also include a great scripting tool called KiXtart. This tool is a logon script processor and an enhanced batch scripting language developed for Windows by Ruud van Velsen of Microsoft Netherlands. It's free software, but the author calls it CareWare and asks that users donate to a charity in lieu of paying a fee for using it.

You will find documentation for the tool, as well as the actual files for download, at [kixtart.org](http://kixtart.org). The manual available for download from this site carries 120 pages of extensive documentation.

The only file required for Windows NT and higher is kix32.exe. With it, you can launch any KiXtart script, which is a text file written in the KiXtart scripting language. The language is neither shell scripting nor VBS, but a separate scripting language with a great deal of functionality. It can process commands based on group membership, and it can be used for mapping drives, editing the registry, manipulating the file system, and a lot more.

Each script file, created with a text editor such as Notepad, is saved with the extension .kix. Let's say you create a file and name it myscript.kix. This script can then be run with a simple command like this:

```
Kix32.exe myscript.kix
```

You can execute it at the command prompt or place the command into a Windows shell script.

## End sum

Now that you know about all of these great tool resources available to you, you can start using them in your scripts. You will want to take the time to learn them, of course, as well as practice them extensively on a lab network before you use them on a production network. Fortunately, both the Resource Kits and KiXtart have good documentation on syntax and usage.

# These resources can help you write admin scripts

A lot of excellent references and tutorials are available to help you learn how to write scripts. If you're learning scripting specifically for network administration, however, you should be aware that many resources lean heavily on scripting for the Web. Although the scripting languages may be the same, such as JScript or VBS, their purposes are different. Network administrators will find that resources focusing on Web scripting won't be terribly useful. Thus, I'm going to discuss some of the best books and online resources you can use for further study.

### Useful books

If you go to the [Amazon.com](http://Amazon.com) Web site and search on the keyword *scripting* under books, you'll probably find more than 80 titles listed. Some of them are for UNIX shell scripting, some cover mainly Web scripting, and some deal specifically with scripting for network administration. I'll highlight three of the most useful books that I'm familiar with, but there are certainly other good ones out there.

- [Microsoft Windows Shell Scripting and WSH Administrator's Guide](#), by Jerry Lee Ford, Jr., Premier Press, 2002, ISBN 1-931841-26-8. I really like this one. It covers shell scripting, the Windows Script Host, JScript, and VBS, all from the point of view of a network administrator. It's actually one of the few I've seen that covers using the FileSystemObject, Shell object, and Network object. It gives examples and shows the correct syntax in both JScript and VBS for each one. The section on Windows shell scripting is quite comprehensive.
- [Windows NT Scripting Administrator's Guide](#), by William R. Stanek, M&T Books, 1999, ISBN 0-7645-3309-6. This is similar to the first book, in that it covers using both Windows shell scripting and the Windows Script Host with JScript and VBS. But each book covers various functions in slightly different ways, with different examples. If you're having trouble understanding some aspect of scripting, it can be useful to go to another source. Often, looking at the two books together can lead to a better understanding.
- [Windows Admin Scripting Little Black Book](#), by Jesse M. Torres, The Coriolis Group, 2001, ISBN 1-57610-881-3. This book is quite different from the other two. It is designed to get the network administrator up and running quickly for any given task. Instead of offering a progressive tutorial on scripting, it is divided into specific functions, such as scripting installations and updates, file management, and local and remote system management. Within each functional area, it discusses specific actions, such as scripting a silent Windows 2000 service pack installation, and provides the specific information you need to create that script. It serves as a nice complement to one or both of the first two books.

### Web sites for admin scripters

As you might expect, Microsoft MSDN maintains a Web site for scripting. A good place to start is the [Windows Script](#) page, which can lead you into the details of both JScript and VBS. Although most of this information applies to scripting for the Web rather than network administration, you'll still find some useful information there—such as the detailed documentation on how to use the FileSystemObject object in VBS.

In addition to Microsoft, there are several other good sites that can help you learn Windows scripting. Make sure you take a look at these four:

- [Windows Scripting Solutions](#) (hosted by Windows and .NET Magazine)—First, I should point out that to access the full content of this site, you must subscribe to its periodical, *Windows Scripting Solutions*, for \$129 in the U.S. or \$135 outside the U.S. With that caveat, this site does offer excellent content in the form of useful articles concerning all aspects of scripting. In addition, you can download code to use in your own scripts.
- [LabMice.net](#)—This site is not strictly devoted to scripting but covers all the aspects involved in supporting and working with a Microsoft network. It's free but requests a donation if you find the site useful. The site serves as a knowledge base that indexes Microsoft, third-party articles, and white papers, and it provides links to other sites that host information you might find helpful.



## Windows scripting guide for administrators

- [Windows & .NET Magazine Scripting 101 Series](#)—This four-part series was written several years ago, but it remains an excellent guide for learning the ins and outs of the Windows Script Host and VBS, along with some good code examples.
- [WinGuides Scripting Guide for Windows](#)—This site allows you to access it for free or pay for premium access. It offers reference information for scripting concepts and techniques, including some excellent code samples.

### **Sticking to the script**

Plenty of good information is out there to help you learn and improve scripting skills for network administration, but you need to know how to sort it out from all the information that's geared toward Web scripting. Once you do, the key will be to just roll up your sleeves and get to work on it. Dig in and write scripts and test them in a lab environment—and don't be afraid to make mistakes.

## Listings

### First steps in VBS scripting for administrators

#### Listing A

```
Dim objFSO
Set objFSO = CreateObject("Scripting.FileSystemObject")
If (objFSO.FileExists("C:\MyFolder\MyFile.txt ")) Then
    objFSO.CopyFile "C:\MyFolder\MyFile.txt", "\\Server1\SharedFolder\MyFile.txt"
Else
    WScript.Echo("Unable to locate file")
End if
```

#### Listing B

```
'Script to copy MyFile.txt to network share
'=====
' Instantiate FileSystemObject
'=====
Dim objFSO
Set objFSO = CreateObject("Scripting.FileSystemObject")
'=====
' Test for existence of MyFile.txt and copy if exists
'=====
If (objFSO.FileExists("C:\MyFolder\MyFile.txt ")) Then
    objFSO.CopyFile "C:\MyFolder\MyFile.txt", "\\Server1\SharedFolder\MyFile.txt"
Else
    WScript.Echo("Unable to locate file")
End if
'=====
' Release FileSystemObject
'=====
Set objFSO = Nothing
Wscript.Quit
```

### Improve administration by using the Shell and Network VBS objects

#### Listing A

```
' This script will create a shortcut to Notepad in the user's desktop folder
' *****
' Instantiate the Shell object
' *****
Dim objShell
Set objShell = WScript.CreateObject("WScript.Shell")
' *****
' Reference the target folder
' *****
Dim targetfldr
Set targetfldr = objShell.SpecialFolders("Desktop")
' *****
' Use the WshShell CreateShortcut method to create the shortcut
' and specify the TargetPath property
' *****
Dim shrtcut
Set shrtcut = objShell.CreateShortcut(targetfldr & "\Notepad.lnk")
Shrtcut.TargetPath = "%windir%\notepad.exe"
' *****
' Save the shortcut
' *****
Shrtcut.Save
```

## Related TechRepublic resources: Scripting guide for Windows administrators



### TechRepublic Books and CDs:

[Quick Reference: Server Commands Pak](#)

[Cisco Router Administration Smart Pak](#)

[Network Administrator's Tool Kit](#)

[Network Administrator's Resource Kit](#)

### Downloads:

[Example scripts for Windows Script Files](#)

[AdminRepublic scripting primer](#)

[Windows Script Host ClipText library files](#)

[The Network Address Inventory script](#)

[Network Monitor Capture Utility script](#)

### Articles and Columns:

[Improve efficiency of admin scripts with programming constructs](#)

[Script helps identify processes displayed by Windows Task Manager](#)

[Customize logon scripts with KiXtart 2001](#)

[Check network status with this Perl script](#)

[Protect your scripts with the Script Encoder](#)

TechRepublic communities engage IT professionals in the ultimate peer-to-peer experience, providing actionable information, tools, and member services to help members get their jobs done. TechRepublic serves the needs of the professionals representing all segments of the IT industry, providing information and tools for decision support and professional advice by job function.

**CIO Republic:** Get analysis and insight on e-business, leadership, executive career, business strategy, and technology.

**IT Manager Republic:** Access technology insights, project management and personal management tips, and training resources.

**NetAdmin Republic:** Get tips on Windows, NetWare and Linux/UNIX administration, infrastructure design and network security.

**Support Republic:** Obtain detailed solutions to desktop hardware, software, and end-user support problems.

**IT Consultant Republic:** Find information and advice on client and vendor relations, project management and technology.

### TechRepublic site features

**Free e-newsletters:** Keep up-to-date on any aspect of the IT industry with TechMails--from tech stocks to daily software tips, from IT careers to hot trends--delivered right to your e-mail Inbox.

**Free downloads:** We've collected resources to make your job easier, including ready-to-use IT forms and templates, checklist, tools, executables, Gartner product analysis, and other white papers.

**TechRepublic's books and CDs:** Find the latest books and CDs available about today's critical IT topics including: PC Troubleshooting, VPN, TCP/IP, Windows Client & Server, Cisco and more.

**Discussion center:** Open a discussion thread on any article or column, or jump into pre-selected topics: career, technology, management, and miscellaneous. The fully searchable Discussion Center brings you the hottest discussions and threads, and allows you to sort them by topic and Republic.

**Try our premium subscription product, TechProGuild, free for 30 days.** Our online IT community provides proven, real-world solutions and the latest articles resources, and discussions affecting frontline IT pros. Get access to over 250 full-text IT books, exclusive downloads, in-depth articles on network and system, PC troubleshooting, networking and Cisco infrastructure information, help desk and support, and more!