# CHAPTER 11

# ADVANCED BATCH FILES

**Learning Objectives**

After completing this chapter you will be able to:

1. List commands used in batch files.
2. List and explain batch file rules.
3. Explore the function of the REM, PAUSE, and ECHO commands.
4. Explain the use of batch files with shortcuts.
5. Explain the purpose and function of the GOTO command.
6. Explain the purpose and function of the SHIFT command.
7. Explain the purpose and function of the IF command.
8. Explain the purpose and function of the IF EXIST/IF NOT EXIST command.
9. Explain the purpose and function of the IF ERRORLEVEL command.
10. Explain the purpose and function of writing programs.
11. Explain the purpose and function of the environment and environmental variables.
12. Explain the use of the SET command.
13. Explain the purpose and function of the FOR...IN...DO command.
14. Explain the purpose and function of the CALL command.

**Student Outcomes**

1. Use the ECHO command to place a blank line in a batch file.
2. Use the GOTO command in conjunction with a label to create a loop.
3. Use a batch file with a shortcut.
4. Use the SHIFT command to move parameters.
5. Use the IF command with strings for conditional processing.
6. Test for null values in a batch file.
7. Use the IF EXIST/IF NOT EXIST command to test for the existence of a file or a subdirectory.
8. Use the SET command.
9. Use the environment and environmental variables in batch files.
10. Use the IF ERRORLEVEL command with XCOPY to write a batch file for testing exit codes.
11. Use the FOR...IN...DO command for repetitive processing.
12. Use the CALL command in a batch file.

**Chapter Overview**

You learned in Chapter 10 how to write simple batch files and use replaceable parameters. Some commands allow you write even more powerful batch files that act like sophisticated programs.

This chapter focuses on the remaining batch file commands, which will allow you to write sophisticated batch files. You will further refine your techniques in working with the environment.

# 11.1 Batch File Commands

A quick summary of batch file rules tells us that any batch file must have the file extension of .BAT or .CMD, it must always be an ASCII file, and it must include legitimate commands. In addition, you can use replaceable or positional parameters to create generic batch files. Batch file commands are not case sensitive. You may use any command in a batch file that you can use on the command line, as well as some specific batch file commands. See Table 11.1 for batch file commands.

| Command | Purpose |
|---|---|
| CALL | Calls one batch program from another without causing the first batch program to stop. Beginning with Windows 2000 Professional, CALL accepts labels as the target of the call. |
| ECHO | Displays or hides the text in batch programs while the program is running. Also used to determine whether commands will be "echoed" to the screen while the program file is running. |
| ENDLOCAL | Ends localization of environment changes in a batch file, restoring environment variables to their values before the matching SETLOCAL command. There is an implicit ENDLOCAL at the end of the batch file. |
| FOR | Runs a specified command for each file in a set of files. This command can also be used at the command line. |
| GOTO | Directs the operating system to a new line that you specify with a label. |
| IF | Performs conditional processing in a batch program, based on whether or not a specified condition is true or false. |
| PAUSE | Suspends processing of a batch file and displays a message prompting the user to press a key to continue. |
| REM | Used to document your batch files. The operating system ignores any line that begins with REM, allowing you to place lines of information in your batch program or to prevent a line from running. |
| SETLOCAL | Begins localization of environmental variables in a batch file. Localization lasts until a matching ENDLOCAL command is encountered or the end of the batch file is reached. |
| SHIFT | Changes the position of the replaceable parameter in a batch program. |

*Table 11.1—Batch File Commands*

You have already used ECHO, PAUSE, and REM. You will now learn the remaining batch file commands, which allow you to create complex batch files. Using these commands is similar to using a programming language. Batch files have a limited vocabulary (the commands listed above), a syntax (punctuation and grammar rules), and a programming logic. Batch files are also limited in the kinds of programming they can do. They do not have the power or the flexibility of a "real"

programming language such as Visual Basic or C++. Batch files, however, accomplish many things in the Windows environment.

## 11.2 A Review of the REM, PAUSE, and ECHO Commands

The REM command in a batch file indicates to the operating system that whatever text follows is to be displayed, but only if ECHO has *not* been turned off. If a command follows REM, it will be displayed but not executed. Remarks can be a string of up to 123 characters, and typically they document batch files. Placing REM in front of a command will allow you to execute a batch file or the CONFIG.SYS file without executing the command that follows it. This allows you to disable a line or lines without having to actually delete them.

The PAUSE command stops a batch file from continuing to execute until you press any key. It tells you to press any key to continue, but does not do any ***conditional processing***.

Remember, to interrupt a batch file, you can always press Ctrl + C or Ctrl + Break. Pressing this key combination will interrupt the execution of the batch file. There is one warning—if the batch file has called an external command and the operating system is in the middle of executing a command such as FORMAT or DISKCOPY, it will finish executing the command before exiting the batch file. Ctrl + C stops the execution of the batch file itself—it will not stop the execution of a .EXE or .COM program.

The ECHO command can be used either on a command line or in a batch file. It is a special command that turns on or turns off the echoing of commands to the screen. If you key in the command ECHO by itself on the command line or include it in a batch file, it will return the status of ECHO: either ECHO on or ECHO off. When ECHO is on, all the commands in a batch file are displayed on the screen. When ECHO is off, you see the output of the command, but not the command itself. Normally ECHO is on, which is particularly useful when you want to track the operation of a batch file. However, when a batch file runs successfully, the display of commands can clutter the screen.

In a batch file, if you do not wish to see each command on the screen, you can issue the command ECHO OFF. The batch file commands are not displayed, but any messages that a command such as COPY issues will be displayed, for example, "1 file(s) copied." Depending on your preferences, you can key in ECHO ON or ECHO OFF within the batch file to display or not display the commands. In addition, if you precede ECHO OFF with @, the command following the symbol, ECHO OFF, will not appear on the screen.

## 11.3 Advanced Features of ECHO and REM

There are some interesting features and variations you can implement with both REM and ECHO. One problem with REM is that the operating system recognizes it as a command and must take time to process it. A shortcut is to use a double colon (::) instead of REM in front of a remark, or documentation line. This will save valuable processing time because the operating system treats all lines beginning

with a colon as a label and ignores them unless they are "called" elsewhere in the batch file. This will be further explained later.

When you turn ECHO off, you still get the display of messages such as "1 file(s) copied." Sometimes you do not wish to see messages. You can use redirection with standard output to redirect the output of a command to a device called NUL. As the name implies, NUL means send it to "nothing." When you send the output to NUL, it goes nowhere, and it is not displayed on the screen.

Although redirecting the output of a command to the NUL device will suppress messages such as "1 file(s) copied," it will not suppress a message that is generated by the operating system like "File not found."

You often want to place a blank line in a batch file for aesthetic purposes or to highlight particular commands. There is, of course, no such thing as a blank line. In the word-processing world, when you want a blank line, you press the Enter key, which places a carriage return in the document and prints as a blank line. This does not work in batch files. In a batch file, the operating system simply ignores it when you press Enter. Pressing Enter does not leave a blank line. If you use REM, you will see nothing if ECHO is off. If you place the word ECHO in the batch file, it will report whether ECHO is on or off. An easy method to get a blank line is to key in ECHO followed by a period. There can be no space between ECHO and the period.

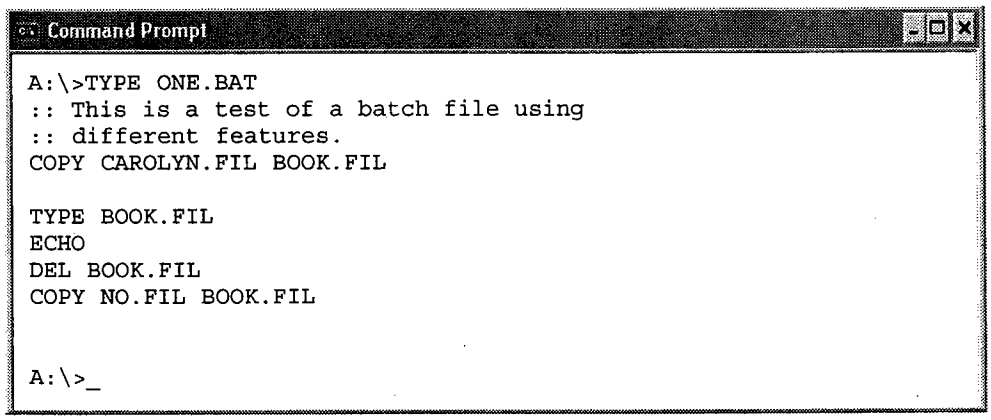
## 11.4 Activity: Using ECHO and NUL

*Note:* The DATA disk is in Drive A. Open a Command Prompt window. A:\> is displayed.

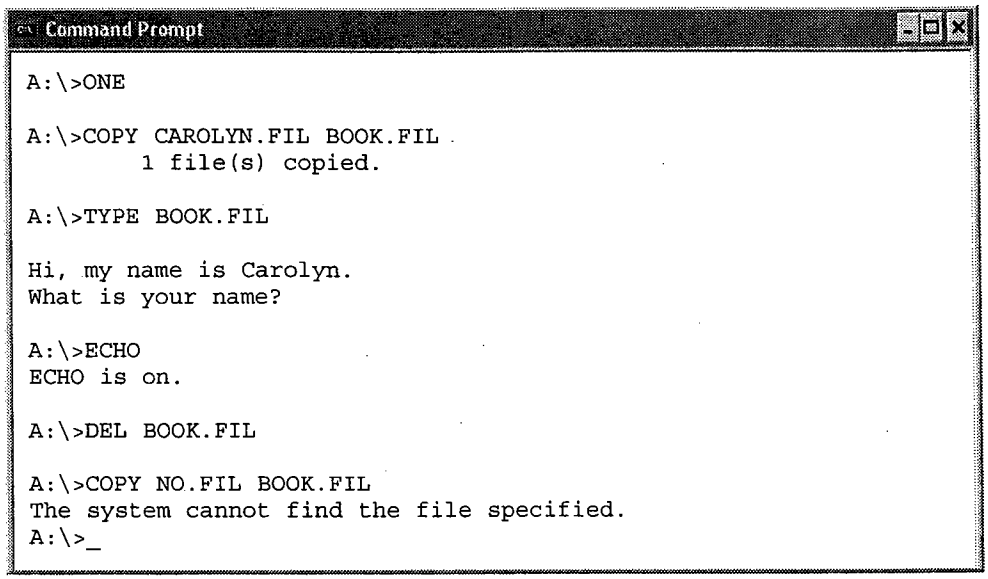
1 Use an editor to create and save the following batch file called **ONE.BAT**, pressing the Enter key only where indicated.

**:: This is a test of a batch file using** Enter
**:: different features.** Enter
**COPY CAROLYN.FIL BOOK.FIL** Enter
Enter
**TYPE BOOK.FIL** Enter
**ECHO** Enter
**DEL BOOK.FIL** Enter
**COPY NO.FIL BOOK.FIL** Enter

2 Close the editor and then key in the following: `A:\>`**TYPE ONE.BAT** Enter



3 Key in the following: `A:\>`**ONE** Enter

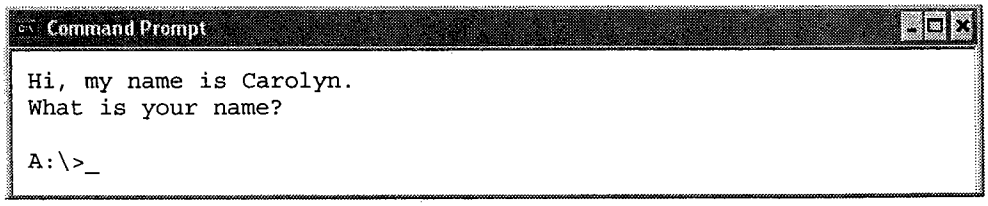



WHAT'S HAPPENING? You see messages as well as the output.

4 Edit and save **ONE.BAT** so it looks as follows:

```
@ECHO OFF
:: This is a test of a batch file using
:: different features.
COPY CAROLYN.FIL BOOK.FIL > NUL
ECHO.
TYPE BOOK.FIL
ECHO.
DEL BOOK.FIL
COPY NO.FIL BOOK.FIL > NUL
```

5 Key in the following: `A:\>`**ONE** Enter




WHAT'S HAPPENING? You now do not see your messages or remarks. You also get some blank lines. The COPY command could not find **NO.FIL** and the error message was redirected to the NUL device and was not displayed to the user on the screen.

## 11.5 The GOTO Command

By using the GOTO command, a batch file can be constructed to behave like a program written in a programming language such as BASIC or C++. The GOTO command will branch to a specific part within a batch file. By using the GOTO statement, you can create a loop. A *loop* is an operation that will repeat steps until

you stop the loop either by using an IF statement or by breaking into the batch file with Ctrl + C.

The GOTO command works in conjunction with a label. This label is not to be confused with a volume label on a disk. A label is any name you choose to flag a particular line, or location in a batch file. A label is preceded by a colon (:) and is ignored by the operating system until called with the GOTO command. A double colon (::), used earlier for REM statements, ensures that the operating system will always disregard the line, even in conjunction with the GOTO statement since a colon may not be used as a label name. A label can be no longer than eight characters. The label itself is not a command, it just identifies a location in a batch file. When a batch file goes to a label, it carries out whatever command follows on the line after the label. GOTO has one parameter—GOTO *label*. Although it is not necessary that labels be exactly the same (i.e., the same case) it is still wise to make them the same case.

## 11.6 Activity: Using the GOTO Command

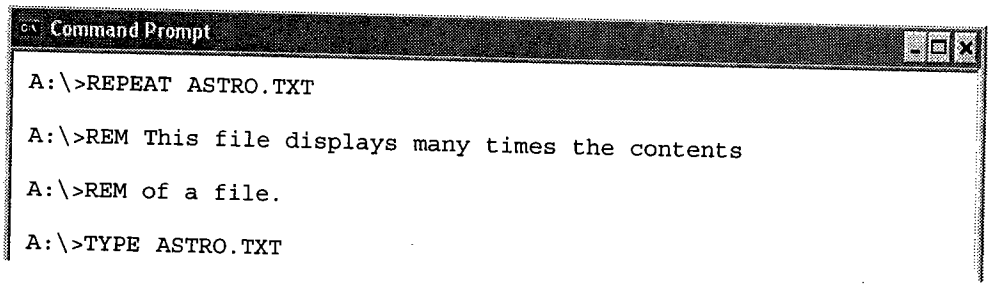
*Note:* The DATA disk should be in Drive A with A:\> displayed.

1 Use any text editor to create and save a batch file called **REPEAT.BAT**. Key in the following using exactly the same case:

**REM This file displays many times the contents** Enter
**REM of a file.** Enter
**:REPEAT** Enter
**TYPE %1** Enter
**PAUSE** Enter
**GOTO REPEAT**

WHAT'S HAPPENING? You have written a small batch file. The first two lines are remarks that will not execute. You are not including ECHO OFF, because you want to see what is happening in your batch file. Omitting ECHO OFF is a way to "debug" a batch file program. *Debug* means to see and repair any errors. The third line (:REPEAT) is a label, which must be preceded by a colon. The fourth line is a simple TYPE command with a replaceable parameter. The PAUSE command is placed on the fifth line so you may see what is happening. The sixth and last line is the loop. The GOTO tells the batch file to return to the label (:REPEAT). It will then return to line 4 and execute the TYPE command. It will then read lines 5 and 6 and continually repeat the process.

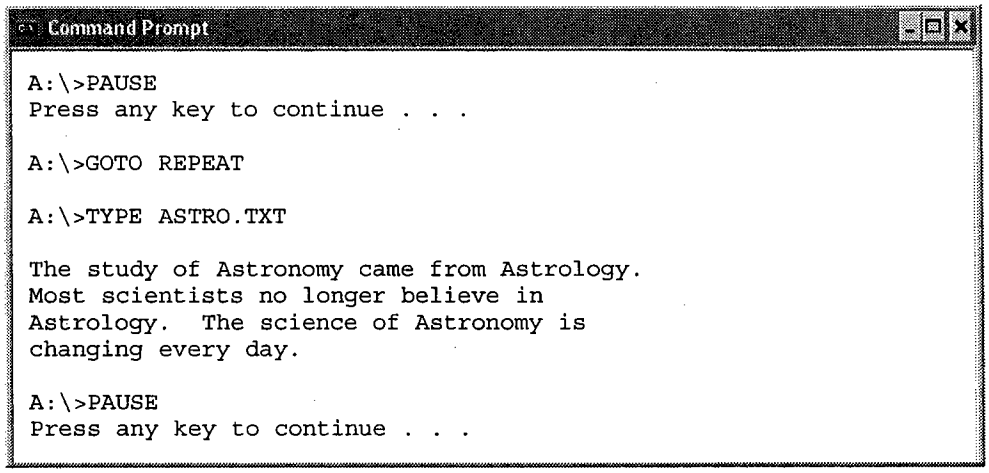
2 You must be at the system prompt, not in the editor. Key in the following:
A:\>**REPEAT ASTRO.TXT** Enter



```
The study of Astronomy came from Astrology.
Most scientists no longer believe in
Astrology.  The science of Astronomy is
changing every day.

A:\>PAUSE
Press any key to continue . . .
```

WHAT'S HAPPENING? The first two lines displayed are remarks. The label was ignored, but the TYPE command was executed. The next line has paused the batch file. You need to press Enter to continue.

3 Press Enter




WHAT'S HAPPENING? The next line in the batch file was GOTO REPEAT. The batch file was sent back to the label :REPEAT. The batch file read the next line after the label, which was the TYPE command, then the next line, which was PAUSE. When you press a key, you will again be returned to the label. Now you see a loop in action.

4 Press Enter a few times to see the loop in action. Then press Ctrl + C to break out of the batch file. It will ask you if you want to terminate the batch job. Key in Y, then Enter for yes. You will be returned to the A:\> prompt.

WHAT'S HAPPENING? A loop can be very useful. For instance, if you wanted to delete all the files from many floppy disks, you could write a batch file that would look like this:

```
@ECHO OFF
:TOP
CLS
ECHO Place the disk with the files you no longer want in
ECHO Drive A.
PAUSE
DEL /Q A:*.*
ECHO Press Ctrl + C to stop executing this batch file.
ECHO Otherwise, press any key to continue deleting files.
PAUSE > NUL
GOTO TOP
```

The /Q parameter makes it so the DEL command does not require a Y or N. You did not want to see the output of the PAUSE command, so you redirected it to the NUL device. Remember, a file like the above which deals with deletion of multiple files can be both convenient and dangerous.

## 11.7 The SHIFT Command

When you have written a batch file with positional parameters, you key in the batch file name followed by a series of values. When the batch file is executed, the operating system looks to the command line for the values it needs to plug into the batch file. It does this based on the position of particular parameters in the command line. In earlier versions of Windows, the TYPE command was limited to one parameter. Thus, if you wanted to use replaceable parameters, you could create a batch file called LIST.BAT whose contents would be:

```
TYPE %1
TYPE %2
TYPE %3
```

Then with this batch file, called LIST.BAT, you would key in the following command line:

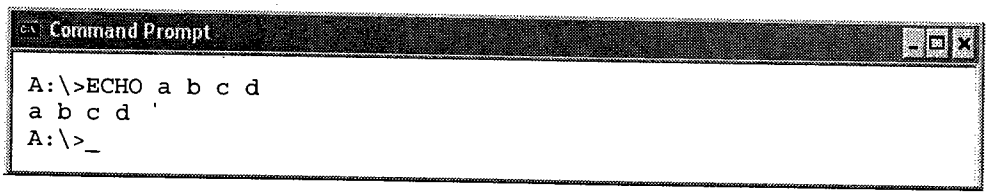
```
LIST APPIL.TXT MAY.TXT JUNE.TXT
```

With this generic batch file, you could key in only three file names. If you wanted more file names, you would have to re-execute the batch file. You are limited to 10 parameters on a command line—%0 through %9. Since %0 actually represents the batch file name itself, you can have only nine parameters. The SHIFT command allows you to shift the parameters to the left, one by one, making the number of parameters on a line limitless. As the SHIFT command shifts the contents of the parameters to the left, parameter 2 becomes parameter 1, parameter 3 becomes parameter 2, and so on. This allows the batch file to process all the parameters on the command line.

## 11.8 Activity: Using the SHIFT Command

*Note:* The DATA disk should be in Drive A with A:\> displayed.

1 Key in the following: `A:\>ECHO a b c d` Enter



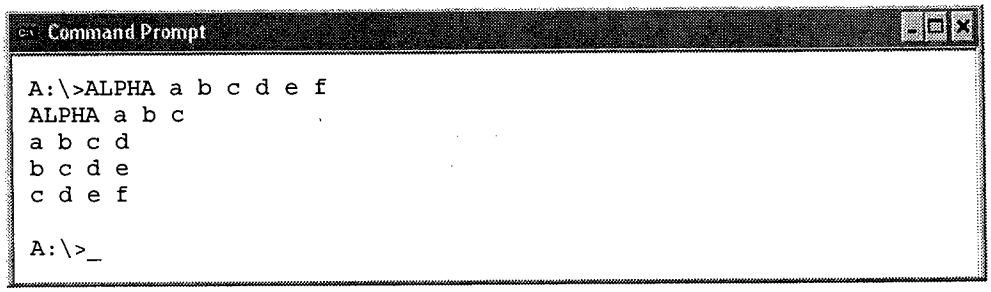

**WHAT'S HAPPENING?** ECHO on a command line just "echoed" what you keyed in. Thus, the parameters a, b, c, and d on the command line were repeated on the screen. If you wanted to display more than five parameters and place the echoing parameters in a batch file, you would need to use the SHIFT command.

2 Use any text editor to create and save the file **ALPHA.BAT** as follows:

```
@ECHO OFF
ECHO %0 %1 %2 %3
SHIFT
ECHO %0 %1 %2 %3
SHIFT
ECHO %0 %1 %2 %3
SHIFT
ECHO %0 %1 %2 %3
```

WHAT'S HAPPENING? You have created a batch file with replaceable parameters. The purpose of the batch file is to demonstrate the SHIFT command. Remember that ECHO just echoes what you keyed in. In your command line, however, even though you have only four parameters (0 through 3), you want to key in more than four values.

3 Remember, you must be at the system prompt, not in the editor. Key in the following: `A:\>`**ALPHA a b c d e f** Enter




WHAT'S HAPPENING? Notice the output. In each case, when the batch file read SHIFT, it moved each parameter over by one position.

| Batch File | Supplied Value from Command Line | Screen Display |
|---|---|---|
| @ECHO OFF | | |
| ECHO %0 %1 %2 %3 | ALPHA is %0<br>a is %1<br>b is %2<br>c is %3 | ALPHA a b c |
| SHIFT | ALPHA is dropped as %0<br>a becomes %0<br>b becomes %1<br>c becomes %2<br>d becomes %3 | |
| ECHO %0 %1 %2 %3 | a is %0<br>b is %1<br>c is %2<br>d is %3 | a b c d |
| SHIFT | a is dropped as %0<br>b becomes %0 | |

| | | |
|---|---|---|
| | c becomes %1<br>d becomes %2<br>e becomes %3 | |
| ECHO %0 %1 %2 %3 | b is %0<br>c is %1<br>d is %2<br>e is %3 | b c d e |
| SHIFT | b is dropped as %0<br>c becomes %0<br>d becomes %1<br>e becomes %2<br>f becomes %3 | |
| ECHO %0 %1 %2 %3 | c is %0<br>d is %1<br>e is %2<br>f is %3 | c d e f |

You should see that you are indeed shifting parameters, but how is this useful? You will write some batch files that use SHIFT so you can see how this technique can be used.

As you know, the operating system stamps each file with the current date and time when it is created or modified. Most often, this means that each file has a unique time and date based on the last time you modified or created the file. Sometimes, you want to place a specific time and date stamp on a file or group of files. For example, if you sell software and you have customers to whom you send files, you might like to ascertain which version of the file they have. By having a particular date and/or time on the file, you can easily keep a date log that is not dependent on the file modification date.

Commands such as XCOPY can back up files after or before a certain date. To ensure that you are backing up all the files you want, you can set the date and update the date stamp on your files. Then you can backup from that date. You need a way to update the dates. You can do this by using the following command:

**COPY *filename* /b +**

Remember, the + sign tells the operating system to concatenate files. The first thing that happens when copying files is a file name is created with the current time and date in the destination directory. At first, the new file is empty. Since there is no specific destination file name, COPY will default to the source file name. It then proceeds to concatenate (add) the existing file to the "new" file name and the new date and time. In essence, it is copying a file onto itself. Since it is a new entry in the directory table, it has the current date and time.

The /B switch tells the operating system to copy the file in binary mode. When you concatenate files with no switches, the files are copied in text mode. The COPY command knows the contents of the file have ended when it sees a special mark called an ***EOF (end-of-file) mark.*** Typically, the EOF mark is **Ctrl** + **Z**. The instant COPY sees this special signal, it thinks there is no more information to copy and will place its own EOF mark at the end of the file—another **Ctrl** + **Z**. Unfortunately, this "extra" EOF mark is sometimes interpreted by a program or a data file as something

other than the end of the file. Thus, you could be in the situation of not copying the entire file. The alternative is to copy the file in binary mode. When you choose this option—the /B switch—COPY will not read the file but will copy everything in the file, ensuring that the entire file contents are copied without adding an extra Ctrl + Z. An extra Ctrl + Z can create problems when you are trying to use the copied file.

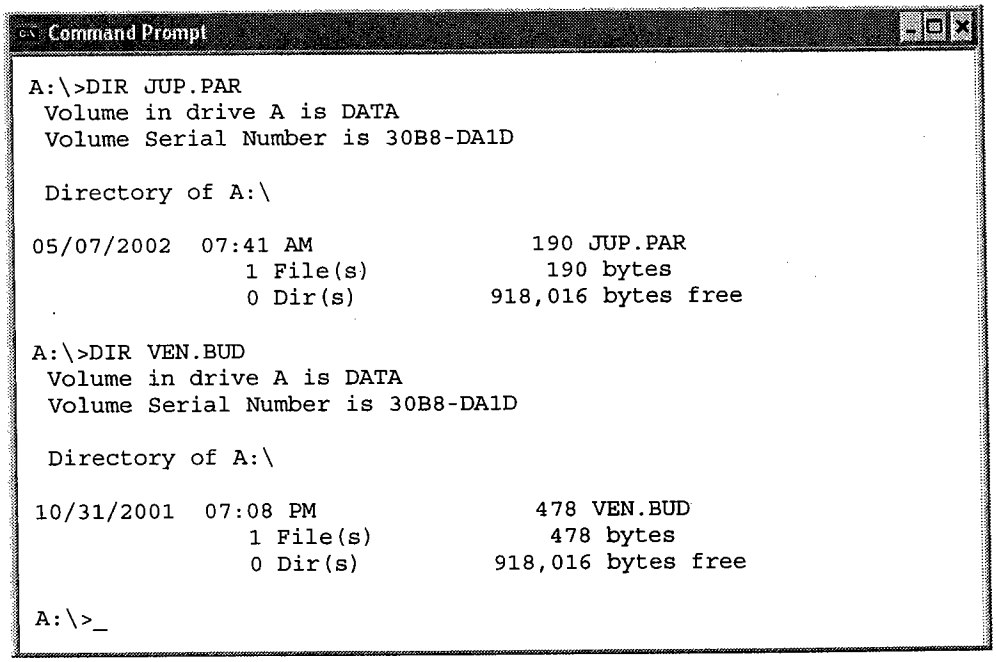
Now that you know how and why to update file dates and times, it is easy to place these commands in a batch file. Since you may have more than one file you wish to "stamp," you want to allow for many file names by using the SHIFT command.

4 Use any text editor to create and save a new file. Name the file **UPDATE.BAT** and then key in the following:

**:DOIT** Enter
**COPY %1 /B + > NUL** Enter
**SHIFT** Enter
**PAUSE** Enter
**GOTO DOIT**

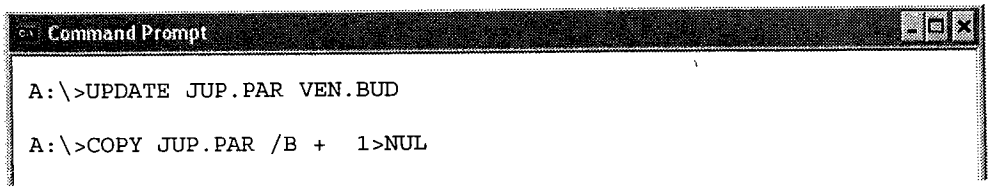
5 Key in the following: `A:\>`**DIR JUP.PAR** Enter

6 Key in the following: `A:\>`**DIR VEN.BUD** Enter




**WHAT'S HAPPENING?** You want the files **JUP.PAR** and **VEN.BUD** to have today's date and time on them. Remember, the number of bytes free on your DATA disk will not necessarily match the number shown in the directory displays in the book.

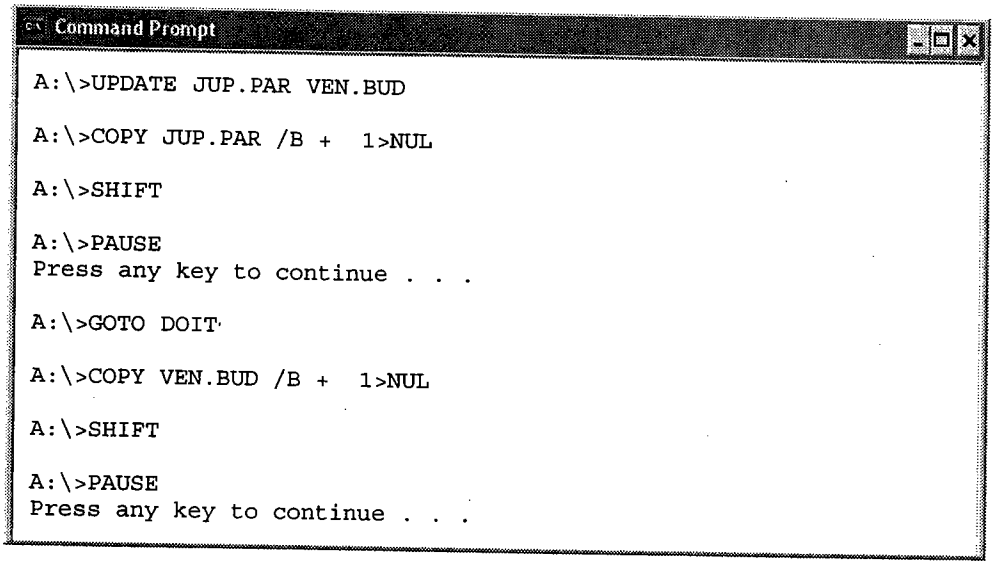
7 Key in the following: `A:\>`**UPDATE JUP.PAR VEN.BUD** Enter



```
A:\>SHIFT

A:\>PAUSE
Press any key to continue . . .
```

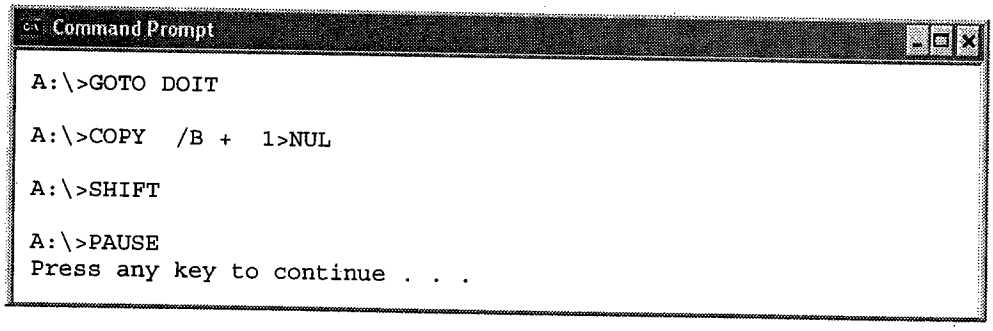
**WHAT'S HAPPENING?** The batch file copied **JUP.PAR**, went to SHIFT, and is now going to copy the next parameter it shifted, **VEN.BUD**.

**8** Press Enter




**WHAT'S HAPPENING?** It copied **VEN.BUD**.

**9** Press Enter

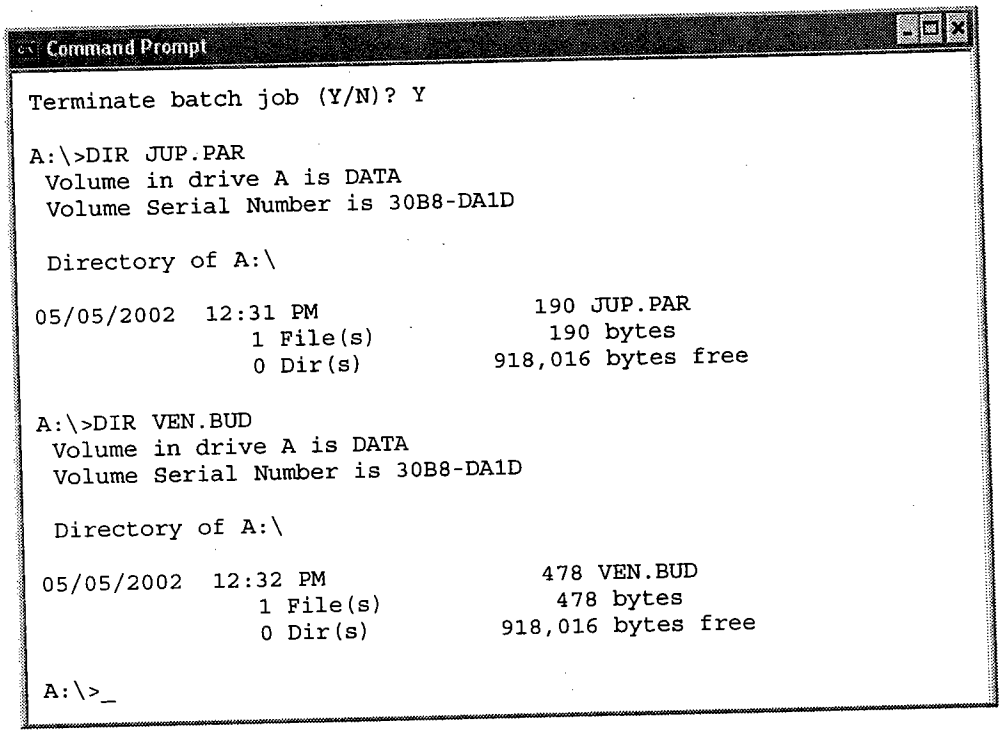



**WHAT'S HAPPENING?** This batch file seemed to work effectively when you first initiated it. It copied **JUP.PAR**. It then shifted over to **VEN.BUD** and copied that. However, you created an endless loop. When the batch file finished copying **VEN.BUD**, it again shifted parameters, but there was nothing to shift to. The batch file is continually going to the label and then trying to execute the command. There is something missing here: a condition that you need to insert. First, though, you must break into the batch file.

**10** Press Ctrl + **C** and answer **Y** to the prompt.

**11** Key in the following: `A:\>`**DIR JUP.PAR** Enter

**12** Key in the following: `A:\>`**DIR VEN.BUD** Enter




**WHAT'S HAPPENING?** The file worked—the date and time should be the current date and time. You can create a batch file and use SHIFT to identify the size and number of files in a directory so you can determine whether the files will fit on a floppy disk.

**13** Use any text editor to create and save a file named **SIZE.BAT** that contains the following:

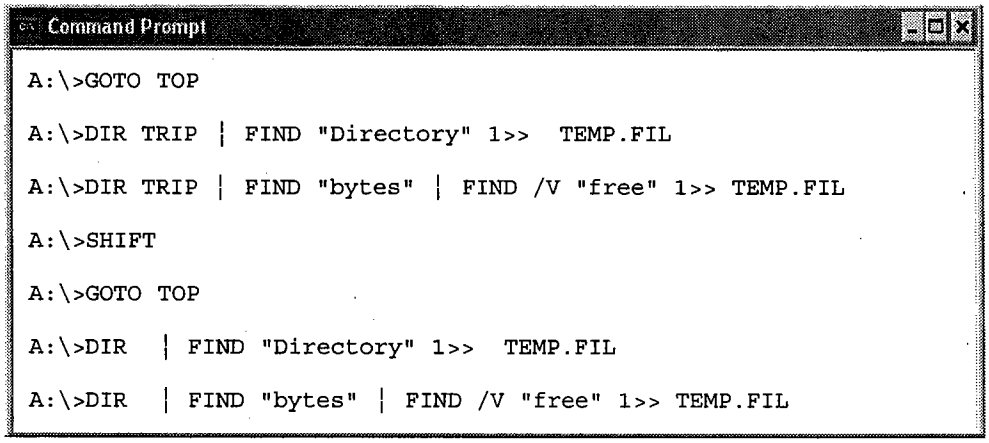
```
:TOP
DIR %1 ¦ FIND "Directory" >> TEMP.FIL
DIR %1 ¦ FIND "bytes" ¦ FIND /V "free" >> TEMP.FIL
SHIFT
GOTO TOP
TYPE TEMP.FIL
PAUSE
DEL TEMP.FIL
```

**WHAT'S HAPPENING?** Since you do not care about the names of the files, only the size and the directory they are in, you filtered the output from the DIR command to include only the items that you wanted. In a normal directory display such as the one shown below, you want to capture the highlighted lines:

```
 Volume in drive C has no label.
 Volume Serial Number is 07D1-080F

 Directory of C:\WUGXP

10/30/2001  01:46p                148 AST.99
10/31/2001  07:08p                478 VEN.99
10/30/2001  03:42p                190 JUP.99
10/31/2001  01:08p                406 MER.99
               4 File(s)          1,222 bytes
               0 Dir(s)  68,145,774,592 bytes free
```

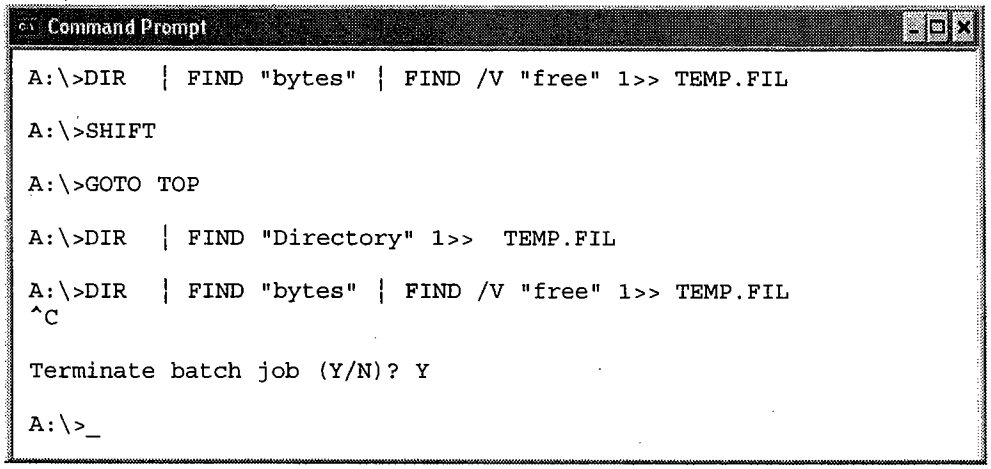
In order to do so, you had to filter the output. The line in the batch file, **DIR %1 ¦ FIND "Directory" >> TEMP.FIL**, found the first highlighted item. It is the only line with "Directory" in it. The line in the batch file, **DIR %1 ¦ FIND "bytes" ¦ FIND /V "free" >> TEMP.FIL**, was looking for a line with the word "bytes" in it. There are two lines with "bytes." You only want the first line, 4 File(s) 1,222 bytes, so you piped the output eliminating the other line (/V) which removed any line with the word "free" in it (0 Dir(s) 68,145,774,592 bytes free ). You then used >> so that you would see both the name of the directory and the bytes in the directory. Had you not used >>, you would have *overwritten* **TEMP.FIL**. At the end of your work, delete **TEMP.FIL** so it will not take space on your disk.

**14** Key in the following: `A:\>`**SIZE CLASS TRIP** Enter



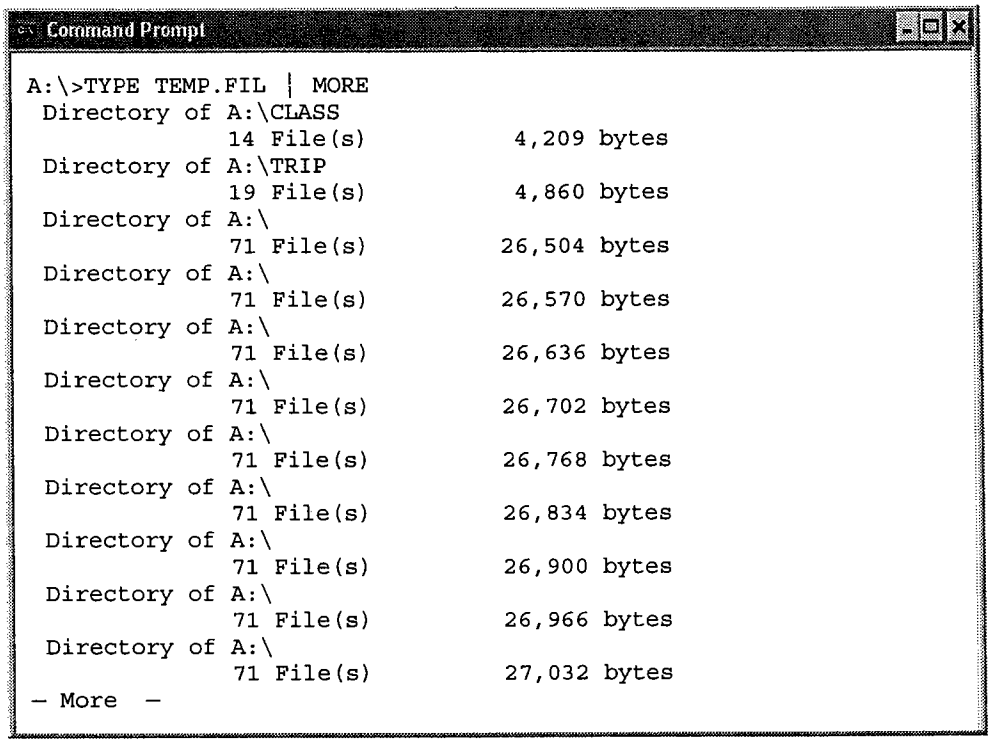

**WHAT'S HAPPENING?** Your batch file is running endlessly. You again created an endless loop.

**15** Press Ctrl + Break and answer **Y** to the prompt.




**WHAT'S HAPPENING?** Your display may look different depending on where you broke into the batch file.

**16** Key in the following: `A:\>`**TYPE TEMP.FIL ¦ MORE** Enter




**17** Press Ctrl + **C** to stop the processing, if necessary.

**WHAT'S HAPPENING?** Your file may be shorter or longer, depending on the length of time before you "broke out" with Ctrl + Break. On some systems, Ctrl + C will stop this file from executing, and on other systems, Ctrl + Break is necessary. In any case, you got more information than you wanted. You now know the size of the **CLASS** and **TRIP** directories, but the other information is useless. What you are missing is conditional processing. (The size of your directories may be different, depending on the work you have done on your DATA disk.)

## 11.9 The IF Command

The IF command allows for conditional processing. Conditional processing is a powerful tool in programming. Conditional processing allows a comparison between two items to determine whether the items are identical or whether one is greater than another. A comparison test will yield one of only two values—true or false. If the items are identical, the condition is true. If the items are not identical, the condition is false. Once you establish a true or false value, you can then direct the program to do something based on that value. Conditional processing is often expressed as IF the condition is true, THEN do something; IF the condition is false, THEN do nothing.

In batch files, the IF command will test for some logical condition and then, if the condition is true, the batch file will execute the command. If the test is false, the command will not be executed and the batch file will fall through to the next command line in the batch file. The IF command in batch file processing can check for three conditions:

1. Whether two sets of characters are or are not identical. The characters are called a string, as in a string of data (sometimes referred to as a character string).
2. Whether or not a file exists.
3. The value of the variable in ERRORLEVEL. ERRORLEVEL is a number that a program can set depending on the outcome of a process, such as checking a true/false condition. ERRORLEVEL can check that number.

Here is the syntax for IF/IF NOT ERRORLEVEL (for complete syntax, see Appendix H):

```
Performs conditional processing in batch programs.

IF [NOT] ERRORLEVEL number command
IF [NOT] string1==string2 command
IF [NOT] EXIST filename command

  NOT                 Specifies that Windows XP should carry out
                      the command only if the condition is false.

  ERRORLEVEL number   Specifies a true condition if the last program
                      run returned an exit code equal to or greater
                      than the number specified.

  string1==string2    Specifies a true condition if the specified text
                      strings match.

  EXIST filename      Specifies a true condition if the specified
                      filename exists.
```

## 11.10 The IF Command Using Strings

You can use the IF command with character strings to test whether or not one string is exactly the same as another. You can tell the IF statement to GOTO a label or to perform an operation when the strings match and the condition is true. Conversely, you can tell the IF statement to GOTO a label or perform an operation when the strings do *not* match and the condition is false. What is to be compared is separated by two equal signs (==).

## 11.11 Activity: Using the IF Command with Strings

*Note:* The DATA disk should be in Drive A. The default drive and directory should be A:\>.

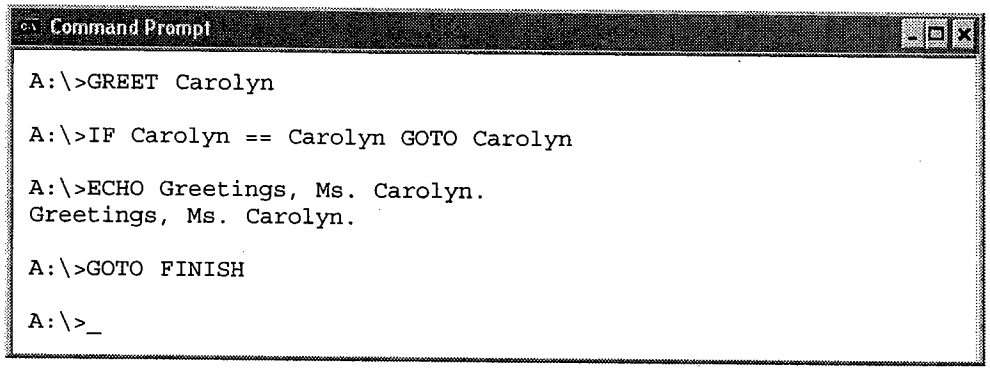
1 Use any text editor to create and save a file called **GREET.BAT**. (*Note:* There are no spaces between the two equal signs.) Key in the following:

**IF %1==Carolyn GOTO Carolyn** Enter
**IF %1==Bette GOTO Bette** Enter
**ECHO Isn't anyone there?** Enter
**GOTO FINISH** Enter
**:Carolyn** Enter
**ECHO Greetings, Ms. Carolyn.** Enter
**GOTO FINISH** Enter

**:Bette** Enter
**ECHO Greetings, Ms. Bette.** Enter
**:FINISH**

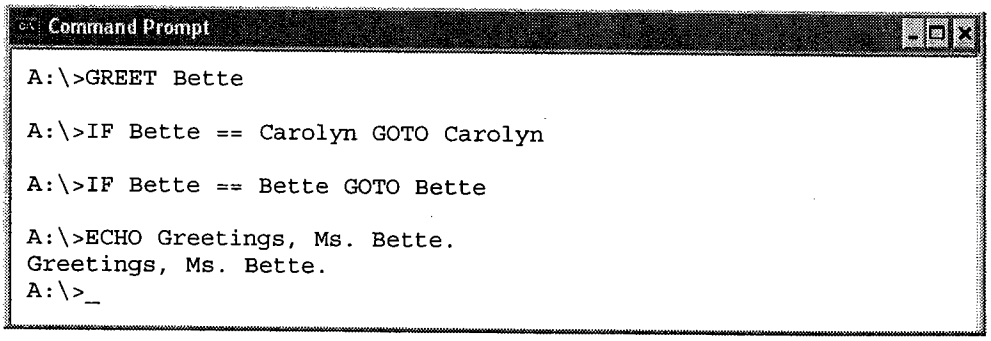
WHAT'S HAPPENING? You have created a batch file to test the IF statement using character strings. You did not place ECHO OFF at the beginning of the file so you can see what happens when it executes.

2 Key in the following: `A:\>`**GREET Carolyn** Enter



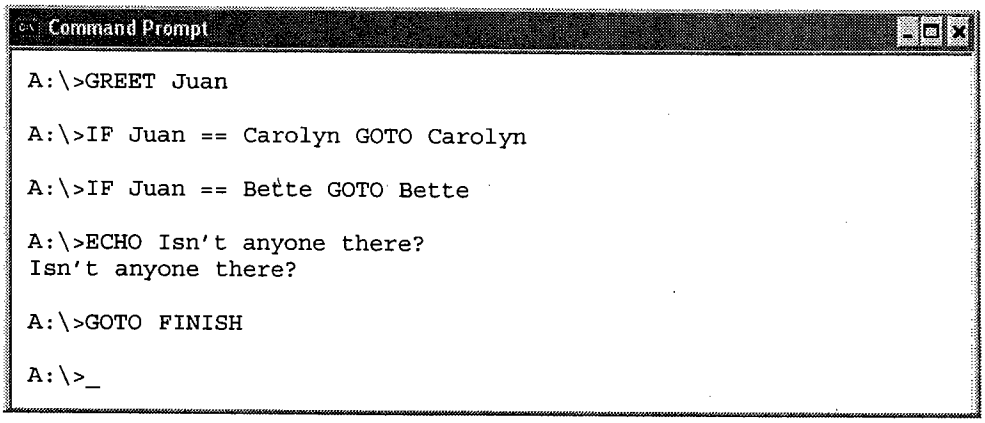

WHAT'S HAPPENING? You keyed in **GREET Carolyn**. The first line in the batch file was executed. When Carolyn took the place of %1, the line read **IF Carolyn==Carolyn**, which is a true statement because the strings of data matched exactly. Since it is true, it performed the GOTO Carolyn command. The line after the label **:Carolyn** was then displayed: **Greetings, Ms. Carolyn.** The line following said **GOTO FINISH**, which it did. After the label **:FINISH**, there were no more lines, and you were returned to the system prompt.

3 Key in the following: `A:\>`**GREET Bette** Enter



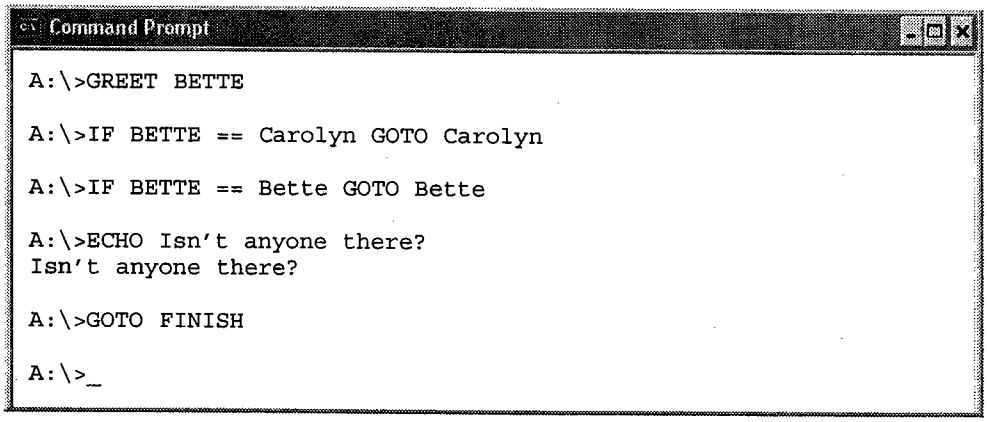

WHAT'S HAPPENING? When you keyed in **GREET Bette**, it read the first line as **IF Bette==Carolyn GOTO Carolyn**. Bette does not equal Carolyn, so it is a false statement. Therefore, the batch file did not go to the label **:Carolyn** but fell through to the next line. The line then read as **IF Bette==Bette**, which is a true statement because the strings of data match exactly. Since it is true, it performed the GOTO Bette command. The line after the label **:Bette** was then displayed: **Greetings, Ms. Bette.** The line following said **:FINISH**, which it did. After the label **:FINISH**, there were no more lines, and you were returned to the system prompt.

4 Key in the following: `A:\>`**GREET Juan** Enter




**WHAT'S HAPPENING?** You keyed in **GREET Juan**. It read the first line as **IF Juan==Carolyn GOTO Carolyn**. Juan does not equal Carolyn, so it is a false statement. The batch file did not go to the label **:Carolyn** but fell through to the next line. The line then read as **IF Juan==Bette**. This is another false statement, so the batch file did not go to the label **:Bette** but fell through to the next line. The line following said **ECHO Isn't anyone there?** Thus, **Isn't anyone there?** was displayed (echoed) to the screen. It then fell through to the next line, which was **GOTO FINISH**. After the label **:FINISH**, there were no more lines, and you were returned to the system prompt.

**5** Key in the following: `A:\>`**GREET BETTE** Enter




**WHAT'S HAPPENING?** You keyed in **GREET BETTE**. It read the first line as **IF BETTE==Carolyn GOTO Carolyn**. BETTE does not equal Carolyn, so it is a false statement. The batch file did not go to the label **:Carolyn** but fell through to the next line. The line then read **IF BETTE==Bette**, which is another false statement. Even though the word is the same, the case is different. Both sides of == must match *exactly*. Because it was not an exact match, the batch file did not go to the label **:Bette**, but fell through to the next line. The line following said **ECHO Isn't anyone there?** Thus, **Isn't anyone there?** was displayed (echoed) to the screen. It then fell through to the next line, which was **GOTO FINISH**. It did. After the label **:FINISH**, there were no more lines, and you were returned to the system prompt. If you wish to ignore case, you can add a parameter, the /I, which when included, tells the batch file to ignore case. The command would be written as

**IF /I %1= =Carolyn GOTO Carolyn**
**IF /I %1= =Bette GOTO Bette**

and so on. The /I must immediately follow the IF statement.

# 11.12 Testing for Null Values

In the above example, you tested for an exact match of character strings. What if you have nothing to test for? For example, in the batch files you wrote, UPDATE.BAT and SIZE.BAT, you used SHIFT. SHIFT kept shifting parameters until all of them were used. When there were no more parameters, you were in an endless loop. You can test to see if a string matches, but what if nothing is there? This is called testing for a ***null value***. You are literally testing for nothing. You must have "something" to test for "nothing." Thus, you place a value in the test that will give you nothing.

There are a variety of methods for testing for null values. One method is to use quotation marks so that your statement becomes IF "%1"=="" GOTO LABEL. The second set of quotation marks is keyed in with no spaces. This statement says, "If nothing is there, GOTO somewhere else." You may also make the line read IF %1void==void GOTO LABEL. If you keyed in GREET Carolyn, your line would then look like Carolynvoid==void. This is not true, so it would proceed to the next line. If there was no value, your line would look like void==void. Now this is true, and the GOTO label would execute. You may use any word; "void" was used in this example. Another method is to use \ so that the statement would become IF \%1\==\\ GOTO LABEL. If you keyed in GREET Carolyn, your line would then look like \Carolyn\==\\. This is not true, so it would proceed to the next line. If there were no value, your line would look like \\==\\. Now this *is* true and the GOTO label would execute.

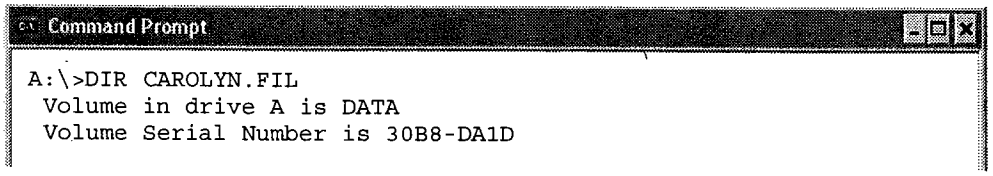
# 11.13 Activity: Using Null Values

*Note:* The DATA disk should be in Drive A. The default drive and directory should be A:\>.

1 Edit and save the file called **UPDATE.BAT** to look as follows:

```
:DOIT
IF "%1"=="" GOTO END
COPY %1 /B + > NUL
SHIFT
PAUSE
GOTO DOIT
:END
```

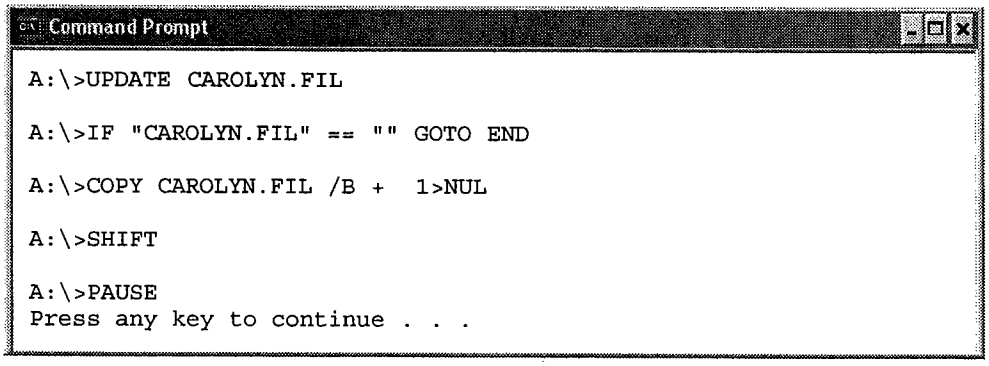
2 Key in the following: `A:\>`**DIR CAROLYN.FIL** Enter



```
 Directory of A:\

07/31/1999  12:53 PM                47 CAROLYN.FIL
               1 File(s)             47 bytes
               0 Dir(s)         915,968 bytes free

A:\>_
```

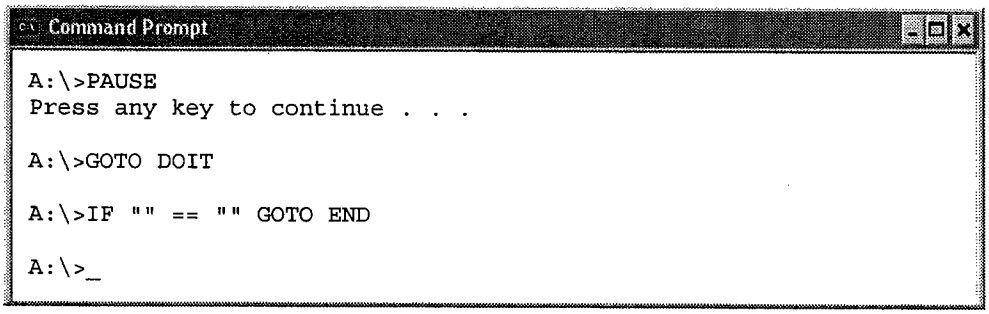
**WHAT'S HAPPENING?** The file called **CAROLYN.FIL** has a date of 7-31-99. You are going to update only the date on this file. SHIFT will still work and will shift "nothing" to %1, but now you are testing for a null value. Once the file is updated, you will go to :END.

**3** Key in the following: `A:\>`**UPDATE CAROLYN.FIL** Enter



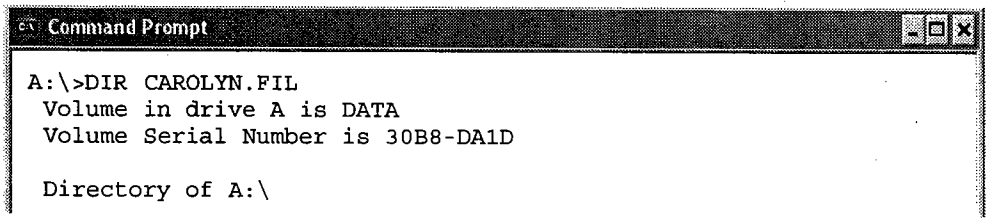

**WHAT'S HAPPENING?** The batch file updated the file **CAROLYN.FIL**. Prior to your testing for a null value, the file looped endlessly. Now you will see if your test for "nothing" works. Remember, there is a SHIFT that will shift over nothing.

**4** Press Enter




**WHAT'S HAPPENING?** Since nothing, or a null value, was there, it was a true condition, and GOTO told it to go to the label called :END. Thus, it skipped the lines and went directly to the end of the batch file.

**5** Key in the following: `A:\>`**DIR CAROLYN.FIL** Enter



```
05/06/2002  11:34 AM                47 CAROLYN.FIL
               1 File(s)             47 bytes
               0 Dir(s)         915,968 bytes free

A:\>_
```

WHAT'S HAPPENING? The date did change to the current date (your date will be different), and you were not in an endless loop. You are now going to try another technique to test for a null value.

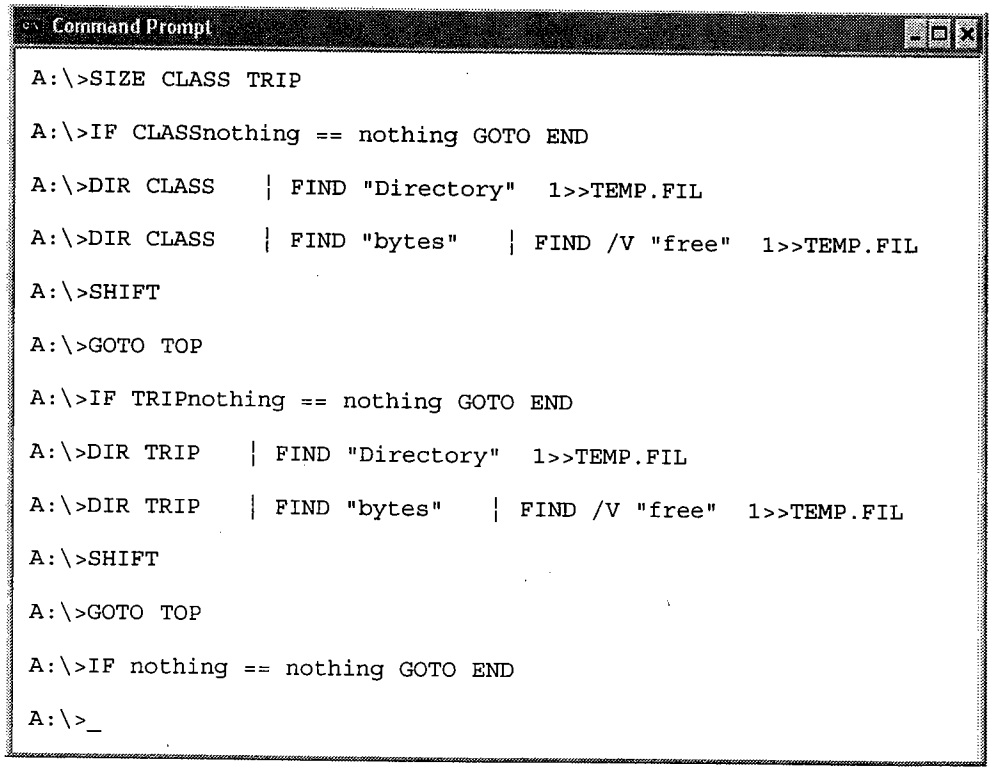
6 Edit and save the file called **SIZE.BAT** to look as follows:

```
:TOP
IF %1nothing==nothing GOTO END
DIR %1 ¦ FIND "Directory" >> TEMP.FIL
DIR %1 ¦ FIND "bytes" ¦ FIND /V "free" >> TEMP.FIL
SHIFT
GOTO TOP
TYPE TEMP.FIL
PAUSE
DEL TEMP.FIL
:END
```

7 Key in the following: `A:\>`**DEL TEMP.FIL** Enter

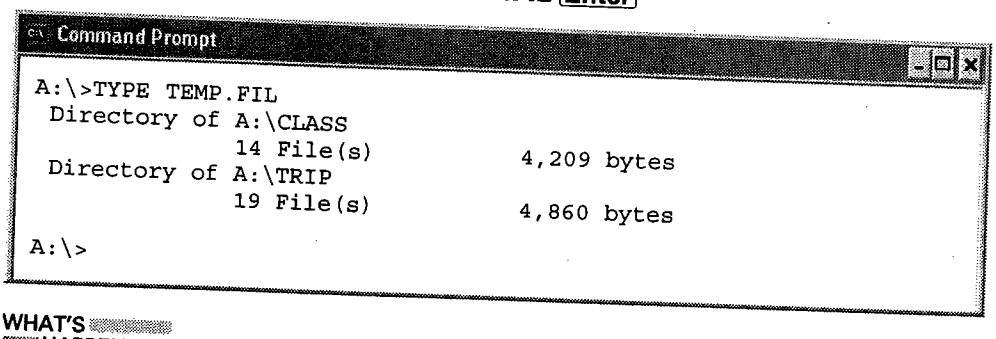
WHAT'S HAPPENING? You wanted to eliminate **TEMP.FIL**, because the last time you ran this batch file, you were stuck in a loop and **TEMP.FIL** did not get deleted. You never reached that line in the batch file.

8 Key in the following: `A:\>`**SIZE CLASS TRIP** Enter



**WHAT'S HAPPENING?** You did not have the problem of an endless loop, but, when you tested for a null value and there was a null value, you told the batch file to GOTO END. It did so, but, by going to the label :END, it never processed the other three lines in the batch file—the lines beginning with TYPE, PAUSE, and DEL. This is why writing batch files (and programs) is a complicated task. You have to think through what you are trying to do and what consequences your instructions will have.

9 Key in the following: `A:\>`**TYPE TEMP.FIL** Enter




**WHAT'S HAPPENING?** The batch file **SIZE.BAT** worked, to some degree. You got the information in the file **TEMP.FIL**, but the file was never displayed or deleted. Thus, you must find another solution to the problem.

## 11.14 The IF EXIST/IF NOT EXIST Command

The IF EXIST command uses a file specification for the test. If the file exists, then the condition is true. Processing then passes to the specified GOTO location or to the command that follows the IF statement. If the file does not exist, the condition is false and the operating system ignores the command in the IF clause. The batch process then reads the next line in the file. When you use IF NOT EXIST and the file does not exist, then the condition is true. Processing then passes to the specified GOTO location or to the command that follows the IF NOT statement. If the file does exist, the condition is false and the batch process will fall through to the next line in the batch file. An important part of the IF EXIST/IF NOT EXIST command is that it works only with file names and *not with directory names.*

## 11.15 Activity: Using IF EXIST to Test for a File

*Note:* The DATA disk should be in Drive A. The displayed prompt is A:\>.

1 Use any text editor to create and save a file called **RENDIR.BAT**. Key in the following:

**IF \%1\==\\ GOTO end** Enter
**IF NOT \%2\==\\ GOTO next** Enter
**ECHO You must include a destination name** Enter
**ECHO for the new directory name.** Enter
**GOTO end** Enter
**:next** Enter
**IF EXIST %1 GOTO message** Enter
**REN %1 %2** Enter

**GOTO end** Enter
**:message** Enter
**ECHO This is a file, not a directory.** Enter
**:end**

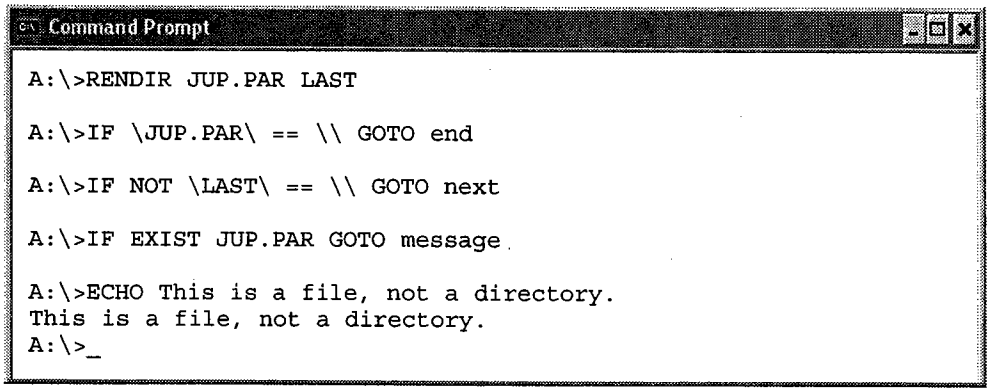
WHAT'S HAPPENING? This batch file will ensure that you are renaming a directory and not a file. The following table analyzes the batch file one line at a time. The line numbers are for purposes of reference only.

1. **IF \%1\==\\ GOTO end**
2. **IF NOT \%2\==\\ GOTO next**
3. **ECHO You must include a destination name**
4. **ECHO for the new directory name.**
5. **GOTO end**
6. **:next**
7. **IF EXIST %1 GOTO message**
8. **REN %1 %2**
9. **GOTO end**
10. **:message**
11. **ECHO This is a file, not a directory.**
12. **:end**

| Batch File by Line Number | Test TRUE | Processing | Test FALSE |
|---|---|---|---|
| 1. IF \%1\==\\ GOTO end | User keys in nothing for %1. Since test is true, action is to go to line 12. | Testing for null value. | User keys in value for %1. Since test is false, action is to go to line 2. |
| 2. IF NOT \%2\==\\ GOTO next | User keys in nothing for %2. Since test is true, action is to go to line 3. | Testing for null value. | User keys in value for %2. Since test is false, action is to go to line 6. |
| 3. ECHO You must include a destination name | | Message for user that he or she did not include a value. | |
| 4. ECHO for the new directory name. | | Continuation of the message. | |
| 5. GOTO end | | Falls through to the GOTO end statement. Action is to go to line 12. | |
| 6. :next | | Label referred to in line 2. | |

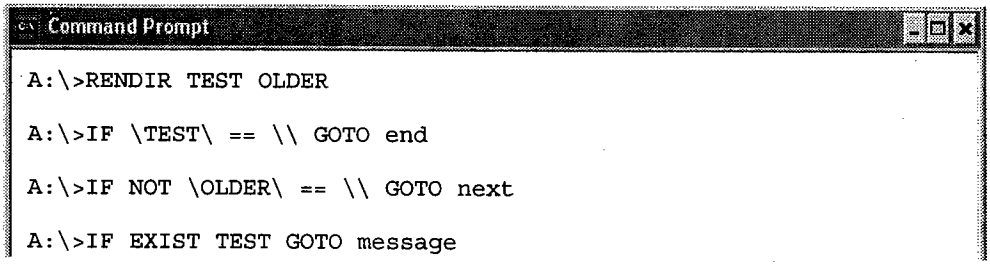
| | | | |
|---|---|---|---|
| 7. IF EXIST %1 GOTO message | User keys in file name for %1. Since test is true, action is to go to line 10. | Testing for value for %1. Is it a file or a directory? | User keys in directory for %1. Since test is false, action is to go to line 8. |
| 8. REN %1 %2 | | Since %1 test is false (not a file), re-naming directory can proceed. | |
| 9. GOTO end | | After directory is renamed, falls through to GOTO end. | |
| 10. :message | | Label referred to in line 2. | |
| 11. ECHO This is a file, not a directory. | | Message that user used a file name, not a directory name. | |
| 12. :end | | | |

**2** Key in the following: `A:\>`**RENDIR JUP.PAR LAST** Enter




**WHAT'S HAPPENING?** Since **JUP.PAR** is a file, the line **IF EXIST JUP.PAR** is true. Since it is true, the batch file executed GOTO and went to the label **:message**. What if it is a directory and not a file?

**3** Key in the following: `A:\>`**RENDIR TEST OLDER** Enter

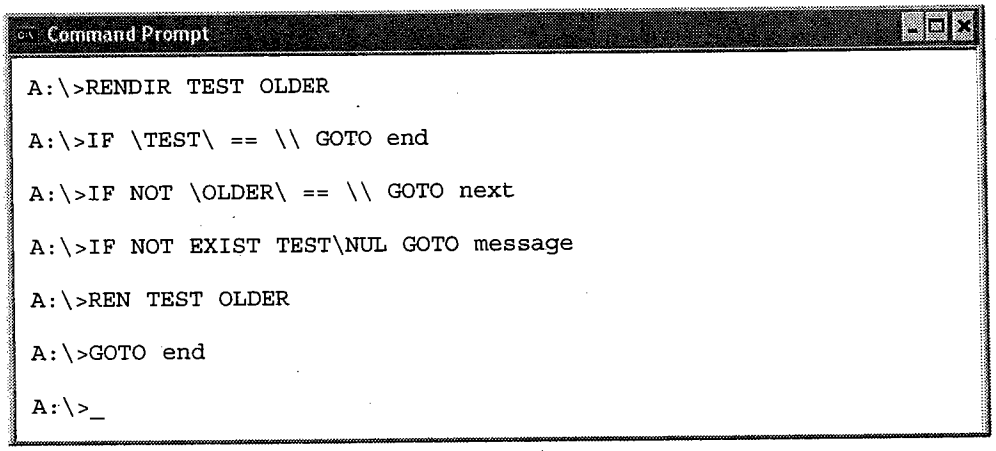


```
A:\>ECHO This is a file, not a directory.
This is a file, not a directory.
A:\>_
```

**WHAT'S HAPPENING?** The if statement identified TEST as a file and therefore went to the message. The directory was not renamed. You cannot use IF EXIST to check for the existence of a directory, as it only works with files. There is a way around this—you can "fool" the IF EXIST command. To check for the existence or nonexistence of a directory, you must use NUL. The null (NUL) device does exist in every directory. NUL is a device that discards anything sent to it. By using %1\NUL, you force IF EXIST/IF NOT EXIST to check for a directory name and not a file name. IF looks for a NUL file (or the nonexistence of a nothing file) in the directory represented by %1. If it cannot get through %1 to look for NUL, then %1 does not exist.

4 Edit the **RENDIR.BAT** file so that you change the line
**IF EXIST %1 GOTO message**
to read
**IF NOT EXIST %1\NUL GOTO message.**

5 Key in the following: `A:\>`**RENDIR TEST OLDER** Enter




**WHAT'S HAPPENING?** The question asked with the IF NOT EXIST statement was testing whether TEST was a directory. If this statement were false (a file name), then the batch file would execute the command following IF (the GOTO message). Since TEST does exist (TEST\NUL) the statement is true, the command following IF is ignored, and the batch file falls through to the next line and renames the directory from TEST to OLDER. Using the logic you just learned, you can correct (debug) **SIZE.BAT** so that it processes all the lines in the batch file. There is one more piece of information you need.

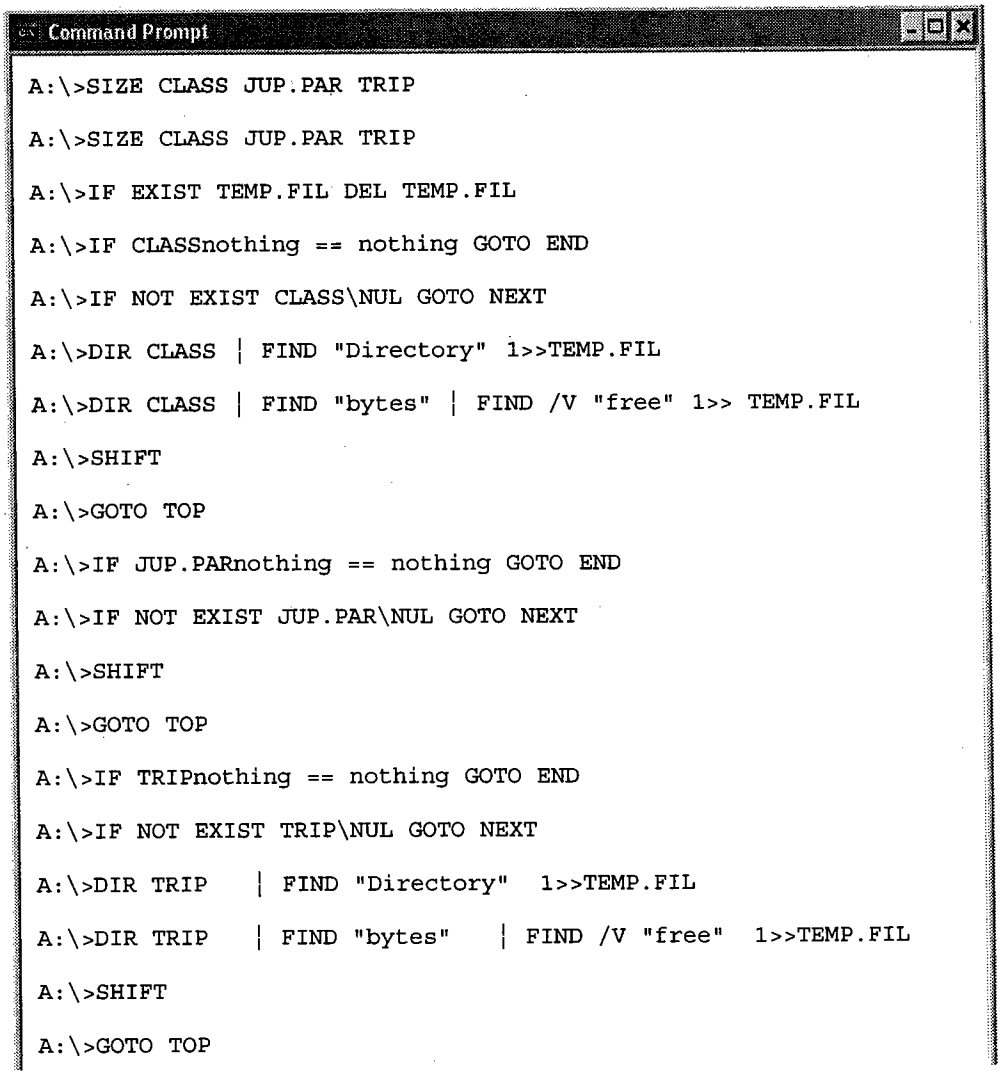
6 Edit and save the file called **SIZE.BAT** to look as follows:

```
IF EXIST TEMP.FIL DEL TEMP.FIL
:TOP
IF %1nothing==nothing GOTO END
IF NOT EXIST %1\NUL GOTO NEXT
DIR %1 ¦ FIND "Directory" >> TEMP.FIL
```

```
DIR %1 ¦ FIND "bytes" ¦ FIND /V "free" >> TEMP.FIL
:NEXT
SHIFT
GOTO TOP
:END
TYPE TEMP.FIL
PAUSE
DEL TEMP.FIL
```

**WHAT'S HAPPENING?** The first line, IF EXIST TEMP.FIL DEL TEMP.FIL, looks for the file called **TEMP.FIL** and delete it if it exists. Then when you create **TEMP.FIL**, it will be a new file every time. The next addition, IF NOT EXIST %1\NUL GOTO NEXT, will see if a directory exists. That is the purpose of %1\NUL. If it is a file, the batch file will go to the :NEXT label, SHIFT, and go back to the :TOP label. The :TOP label is not at the top of the batch file because you want to delete **TEMP.FIL** only the first time you execute the batch file. Notice that you had to move the :END label. In its previous batch file location, you would not have been able to read **TEMP.FIL**.

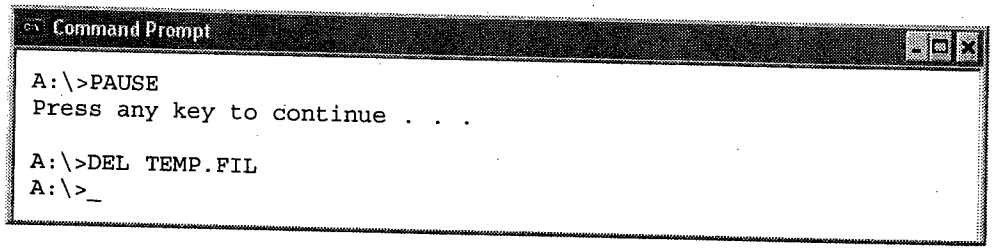
7 Key in the following: `A:\>`**SIZE CLASS JUP.PAR TRIP** Enter



```
A:\>IF nothing == nothing GOTO END

A:\>TYPE  TEMP.FIL
 Directory of A:\CLASS
              14 File(s)          4,209 bytes
 Directory of A:\TRIP
              19 File(s)          4,860 bytes

A:\>PAUSE
Press any key to continue . . .
```

**WHAT'S HAPPENING?** Your batch file worked correctly. It used **JUP.PAR**, knew it was a file, and did not include it in the output. The more complicated you want a batch file to be, the more you will have to analyze the logic of what you want to do and how to accomplish it.

**8** Press **Enter**




**WHAT'S HAPPENING?** The batch file deleted the TEMP.FIL when it completed executing, and you have returned to the system level.

## 11.16 The IF ERRORLEVEL Command Testing

A program can set an ***exit code*** when it finishes executing. A batch file can test this exit code with the IF ERRORLEVEL statement. Actually, the name ERRORLEVEL is a misnomer because the number returned does not necessarily mean there was an error. For instance, the test IF ERRORLEVEL 3 will be true if the exit code is greater than or equal to 3. Thus, an exit code is not tested for a match with ERRORLEVEL, but to determine if it is greater than or equal to it. The test IF ERRORLEVEL 0 will *always* be true since every possible exit code is greater than or equal to 0. The trickiest thing about testing ERRORLEVELs in batch files is that the exit codes must be listed in *descending* order when you use IF ERRORLEVEL and in *ascending* order when you use IF NOT ERRORLEVEL. For instance, COPY will set one of the following exit codes:

```
0  Files were copied without error.
1  No files were found to copy.
```

You can write a batch file testing for exit codes.

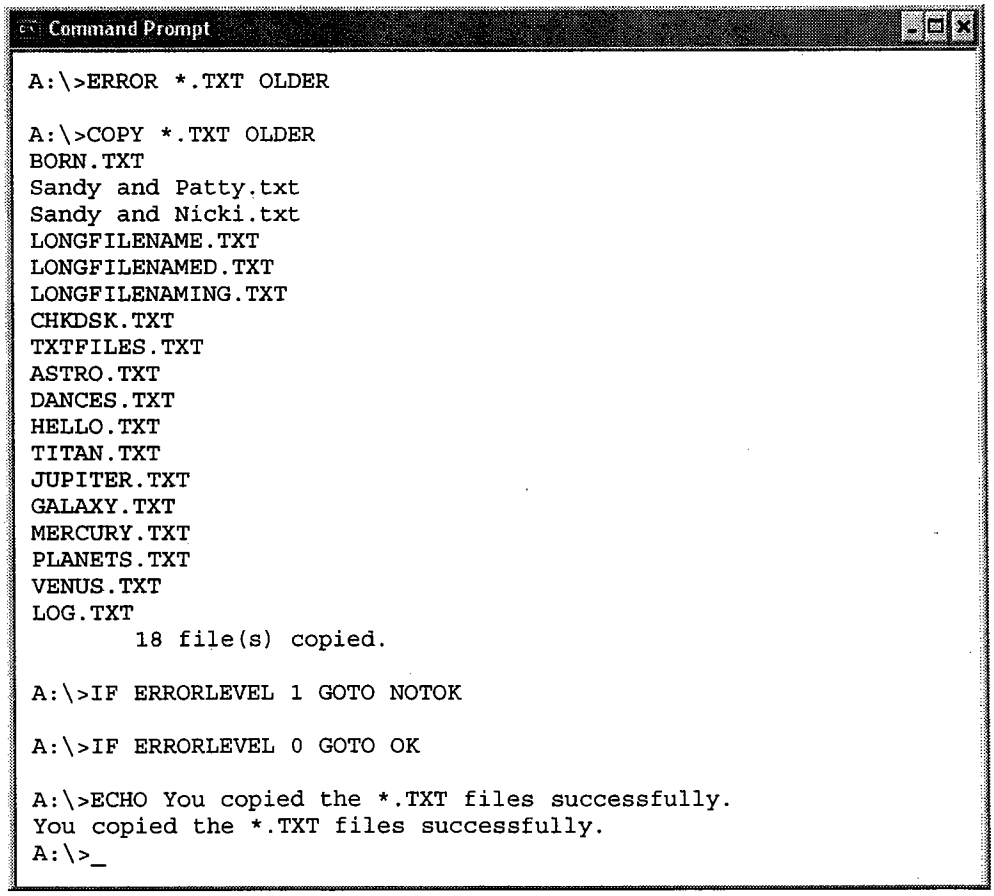
# 11.17 Activity: Using IF ERRORLEVEL with COPY

*Note:* The DATA disk should be in Drive A. The displayed prompt is A:\>.

1 Use any text editor to create and save a file called **ERROR.BAT**. Key in the following:

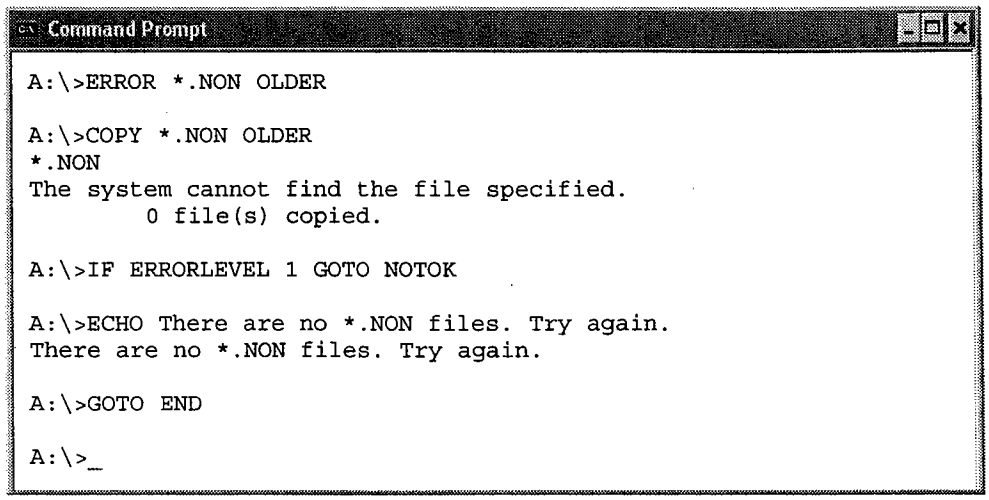
**COPY %1 %2** Enter
**IF ERRORLEVEL 1 GOTO NOTOK** Enter
**IF ERRORLEVEL 0 GOTO OK** Enter
**:NOTOK** Enter
**ECHO There are no %1 files. Try again.** Enter
**GOTO END** Enter
**:OK** Enter
**ECHO You copied the %1 files successfully.** Enter
**:END**

2 Key in the following: A:\>**ERROR *.TXT OLDER** Enter




**WHAT'S HAPPENING?** You successfully copied the **.TXT** files to the **OLDER** subdirectory. The exit code that was generated by COPY gave you the message that the copy was successful.

3 Key in the following: A:\>**ERROR *.NON OLDER** Enter




**WHAT'S HAPPENING?** Again, the exit code was correctly read. As you can see, you can use the exit codes successfully in a batch file. Since programs like COPY give you a message anyway when it could not find the file or files, you may ask yourself, why go to the trouble of writing a batch file? The reason is that you can write a small program to test for other kinds of information.

## 11.18 Writing Programs to Test for Key Codes

Rather than being limited to the exit codes that are set by operating system programs, you can write a small program that will create an exit code based on some activity. For instance, a program can be written that will identify which key was pressed and report which key it was. You can do this because every time you press a key, it is identified by a one- or two-digit ***scan code***. Actually, two things are reported when you press any key on the keyboard. First, that you pressed a key. Second, that you released the key. The keyboard controller tells the CPU that some keyboard activity is occurring. The stream of bytes is converted into the scan code, which identifies the specific key (see Appendix G for a list of scan codes for all the keys).

You are going to write a program that will report the scan code for any key that is pressed on the keyboard. Once you know the reported code, you can test for a specific key using ERRORLEVEL in the batch file. The batch file can then act based on the reported code. In order to do this, you must write a program. Remember, to be executed, a program must be in "bits and bytes"—the 0s and 1s the computer understands.

There are several ways to write a program. One is to know a programming language and be able to turn the programming language program (source code) into executable code (object code). This is called compiling a program—turning a language into code. That task is beyond the scope of this text. Fortunately, there is an easier way that you can create a small program—using an operating system utility program called DEBUG.

DEBUG can directly modify bytes in a file. DEBUG allows you to test and debug executable files—those with a .COM or .EXE file extension. Remember, you cannot

use TYPE to look at a file with the extension of .EXE or .COM because those file extensions indicate programs that are not ASCII-readable files. DEBUG is a small program that has its own commands and syntax. If you know the commands of the DEBUG program and the rules of programming, you could write a .COM program directly with DEBUG. Unless you are a programming expert, you will probably not want to do this.

The easiest way to use DEBUG is to create a script or a ***script file***. A script is a set of instructions that you can write in any ASCII editor. Once you have written the script, you can "feed" it to the DEBUG program via redirection (DEBUG < SCRIPT.FIL). DEBUG will then convert the script file to an executable program with a .COM file extension. Once you have a .COM file, you can execute it as you do any program. This process is the simplest way to create a file that will report the scan code for any key that is pressed. The program you create will be called REPLY.COM.

Since using DEBUG directly can be tricky, the example below shows a .COM program written with DEBUG that will return the scan code of a pressed key. If you want to try to use DEBUG directly, what appears on the screen in this example will be in *this typeface* and what you key in will be in **this typeface**. The hyphen (-) and the colon (:) are prompts presented to you by the DEBUG program. Instructions such as 100 assemble the program at memory address 100 (hexadecimal). 12B3 will vary from machine to machine. In the example shown here, 12B3:0100 represents segment/offset memory address. You must press <enter>after each line and also when <enter>is specified. The following is a summary of commands available within the DEBUG program:

```
assemble     A [address]
compare      C range address
dump         D [range]
enter        E address [list]
fill         F range list
go           G [=address] [addresses]
hex          H value1 value2
input        I port
load         L [address] [drive] [firstsector] [number]
move         M range address
name         N [pathname] [arglist]
output       O port byte
proceed      P [=address] [number]
quit         Q
register     R [register]
search       S range list
trace        T [=address] [value]
unassemble   U [range]
write        W [address] [drive] [firstsector] [number]

allocate expanded memory         XA [#pages]
deallocate expanded memory       XD [handle]
map expanded memory pages        XM [Lpage] [Ppage] [handle]
display expanded memory status   XS
```

The following is shown as an example of how to use DEBUG, but you do not have to do this. If you do, note the differences between the letter l and the number 1. Be sure and check with your lab administrator before attempting to key in this example. Be very sure you are at the A:\> prompt.

A:\>**DEBUG**
**-a 100** Enter
*158E:0100* **mov ah,8** Enter
*158E:0102* **int 21** Enter
*158E:0104* **cmp al,0** Enter
*158E:0106* **jnz 10a** Enter
*158E:0108* **int 21** Enter
*158E:010A* **mov ah,4c** Enter
*158E:010C* **int 21** Enter
*158E:010E* Enter
**-r cx** Enter
*CX 0000*
**:e** Enter
**-n reply.com** Enter
**-w** Enter
*Writing 0000E bytes*
**-q** Enter

An easier way to create REPLY.COM is to create a script file. Again, a script file is merely a text file that contains a series of commands that can be redirected into DEBUG to create a .COM file. The script file is not the program. You use any text editor; name the file, in this case REPLY.SCR; and key in the following commands. Then, to make REPLY.SCR an executable program, you redirect it into DEBUG to create REPLY.COM. The next activity will show you how to create REPLY.SCR and REPLY.COM. (*Note:* You may want to check with your instructor to see if he or she has created REPLY.COM for you.)

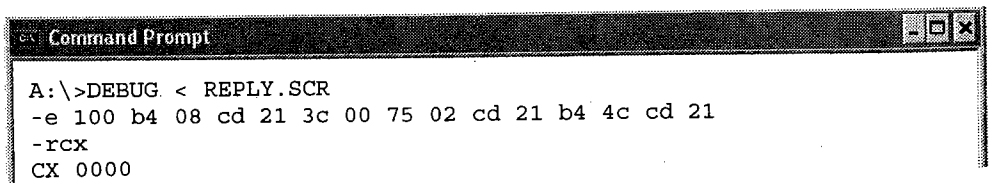
## 11.19 Activity: Writing a Script File

*Note:* The DATA disk should be in Drive A. The displayed prompt is A:\>.

1 Use any text editor to create and save a file called **REPLY.SCR**. Key in the following:
**e 100 b4 08 cd 21 3c 00 75 02 cd 21 b4 4c cd 21** Enter
**rcx** Enter
**e** Enter
**n reply.com** Enter
**w** Enter
**q**

WHAT'S HAPPENING? Now that you have written **REPLY.SCR**, you must now "assemble" it or convert it into the bytes that make it a program. You do this by redirecting the script file into DEBUG.

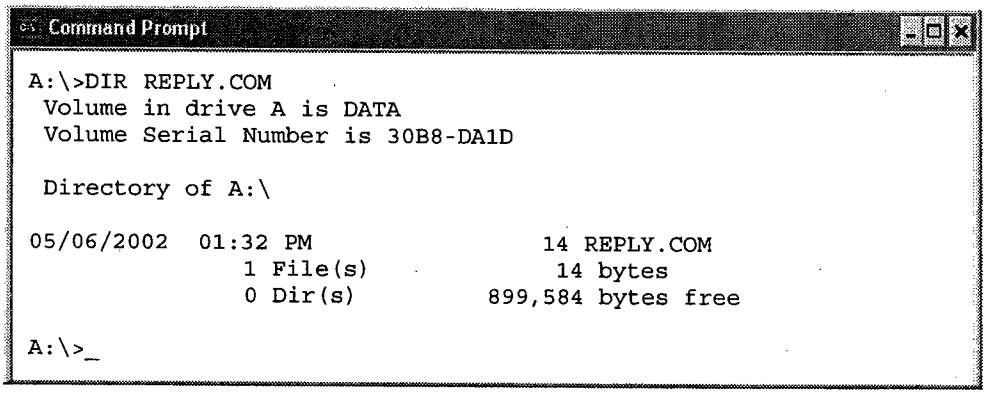
2 Key in the following: A:\>**DEBUG < REPLY.SCR** Enter



```
:e
-n reply.com
-w
Writing 0000E bytes
-q

A:\>_
```

WHAT'S HAPPENING? You have compiled **REPLY.SCR** into a program called **REPLY.COM**.

3 Key in the following: `A:\>`**DIR REPLY.COM** Enter



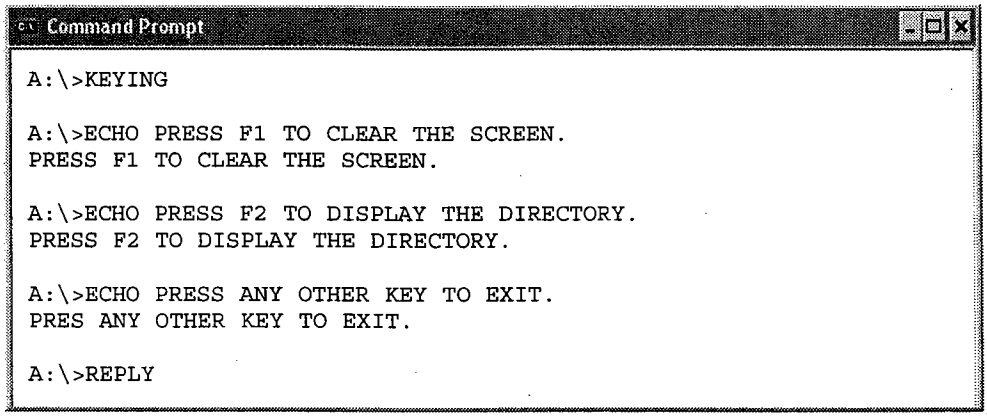

WHAT'S HAPPENING? Now that you have written a program, you want to use it in a batch file.

4 Use any text editor to create and save a file called **KEYING.BAT** that contains the following:

```
ECHO PRESS F1 TO CLEAR THE SCREEN.
ECHO PRESS F2 TO DISPLAY THE DIRECTORY.
ECHO PRESS ANY OTHER KEY TO EXIT.
REPLY
IF ERRORLEVEL 61 GOTO END
IF ERRORLEVEL 60 GOTO F2
IF ERRORLEVEL 59 GOTO F1
GOTO END
:F1
CLS
GOTO END
:F2
DIR
:END
```

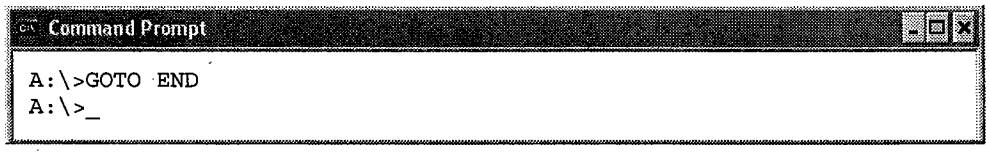
WHAT'S HAPPENING? This is a simple batch file that checks the scan codes you generate by pressing a key. Checking IF ERRORLEVEL codes in descending order is *critical* because the command is tested to determine if the error code is equal to or greater than the value specified. In this program, if you press a key that returns a value of 61 or above, you exit the program. If you press F2, it returns a code of 60. If you press F1, it returns a code of 59. If none of those conditions exist, then you exit the batch file.

**5** Key in the following: `A:\>`**KEYING** Enter




**WHAT'S HAPPENING?** You have executed the **KEYING** batch file. The program called **REPLY.COM** is waiting for you to press a key.

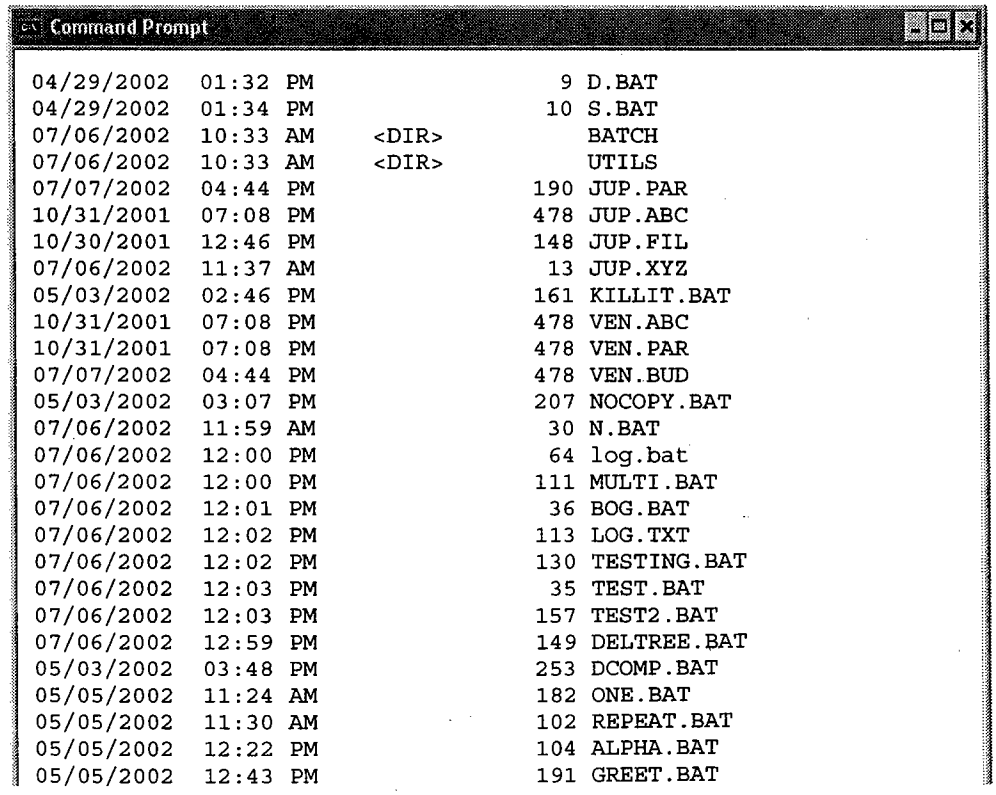
**6** Press F1




**WHAT'S HAPPENING?** Pressing F1 cleared the screen.

**7** Key in the following: `A:\>`**KEYING** Enter

**8** Press F2



```
               77 File(s)         27,184 bytes
               10 Dir(s)         898,560 bytes free
A:\>_
```

**WHAT'S HAPPENING?** (The above graphic represents only a portion of what scrolled by on your screen.) Pressing F2 gave you a directory of your disk. As you can see, **REPLY.COM** checked the scan code returned by the key you pressed and followed the instruction in the batch file based on the key you pressed. Remember, the number of files, directories, and bytes free on your DATA disk will not necessarily match the number shown in the directory displays in the book.

## 11.20 The Environment

The environment is an area that the operating system sets aside in memory. You have used environmental variable PATH in a previous chapter. In Chapter 10, variables that represent the drive and directory used by the operating system were discussed. The environment is like a scratch pad where notes are kept about important items that the operating system needs to know. The environment is like a bunch of post-it notes. Application programs can read any items in the environment and can post their own messages there. A ***variable*** is a value that can change, depending on conditions or on information passed to the program. Data consists of constants or fixed values that never change and variable values that do change. The ***environment*** is, in essence, an area in memory where data can be stored. When evaluating an expression in some environment, the evaluation of a variable consists of looking up its name in the environment and substituting its value. In programming, an ***expression*** is any legal combination of symbols that represents a value. These variables are used by the operating system to discover things about the environment it is operating in. Environment variables can be changed or created by the user or a program.

Programs can get the value of a variable and use it to modify their operation, much like you can use a value in a command line argument. The operating system has the ability to store data in memory. The stored data takes the form of two strings—one is the name of the variable, and the other is the value of the variable. An ***environmental variable*** is a name assigned to a string (value) of data. You can set your own environmental variables. However, there are some common environmental variables that are set when you start Windows. There are environmental variables that are commonly used which usually have short, easy-to-remember names. These environmental variables store information such as your user name (USERNAME); the location where, by default, your files are saved (USERPROFILE); the search path the operating system uses to look for commands (PATH); what is displayed in your prompt (PROMPT); as well as the name of your Windows directory—where the operating system files are kept (SystemRoot). It also includes the location of the file CMD.EXE. You can also leave messages there via batch files or from the command line. You do this with the SET command. Environmental variables set by the operating system will remain in effect throughout the entire work session at the computer. Those set in the Command Prompt window or in batch files executed in the Command Prompt window will remain in effect *only* during that command prompt session. While values are in effect, you can use the syntax

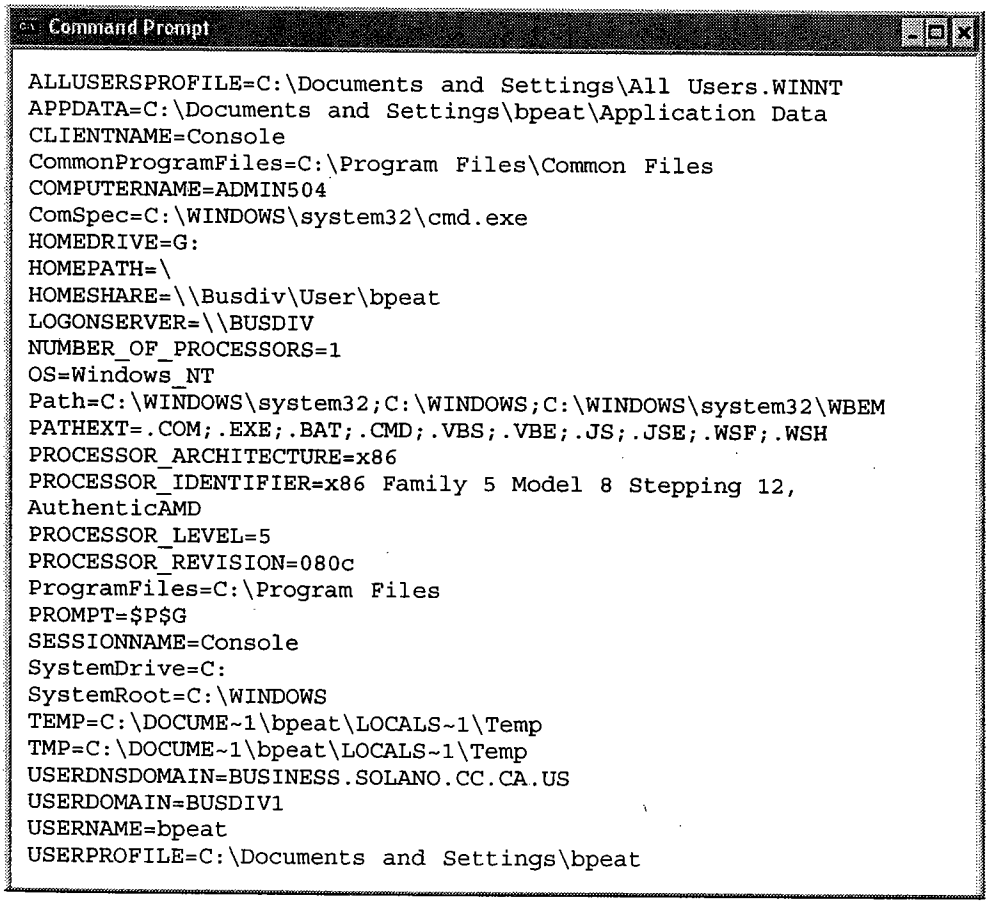
%VARIABLENAME%, which will use the value of the environment variable. To view the value of an environmental variable, you can use the syntax of ECHO %ENVIRONMENTALVARIABLENAME%. The internal command SET allows you to display what is currently in the environment, set environmental variables, or delete environmental variables. If you use the SET command, followed by a letter, the SET command will list any environmental variables that begin with that letter. The basic syntax is:

```
SET [variable=[string]]

  variable          Specifies the environment-variable name.
  string            Specifies a series of characters to assign to the
                    variable.

Type SET without parameters to display the current environment
variables.
```

# 11.21 Activity: Using SET and the Environmental Variables

1 Key in the following: `A:\>`**SET** Enter

*Note:* If the your environment display is too long to fit on one screen, use the MORE filter.

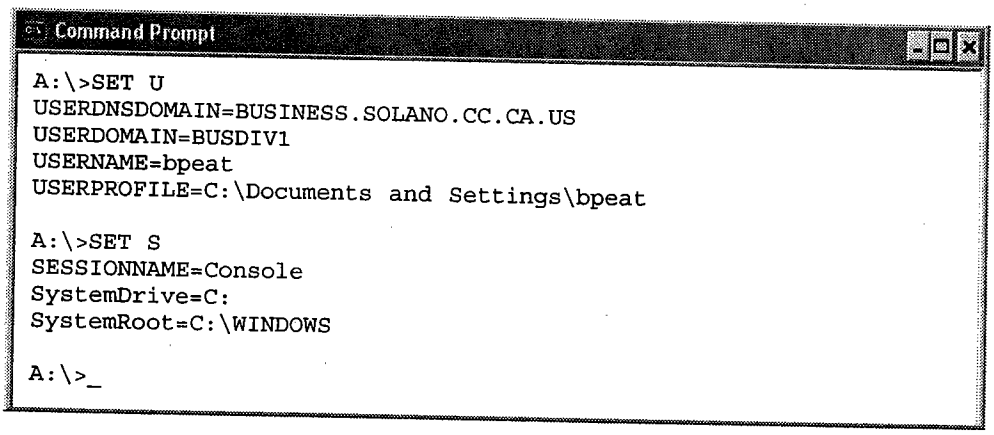


**WHAT'S HAPPENING?** Your values will differ from those shown. As you can see, Windows stores much information about your system (your environment) in the operating system environment. For instance, the environmental variable called ComSpec has a value, in this example, of C:\WINDOWS\system32\cmd.exe. This tells the operating system that the location of CMD.EXE is C:\WINDOWS\SYSTEM32. When you execute a program, Windows no longer needs the command processor in memory. However, when you exit the program and need to key in another command, Windows must reload the command processor from disk. In order to do so, it must know where it is located. It looks up the value of ComSpec to find that location. The PATH value tells Windows what directories and in what order it is to search for executable files. The value for SystemRoot, in this example, is C:\WINDOWS. That tells Windows the name of the directory that holds the Windows operating system files.

2 If you used the MORE filter, press the spacebar until you are returned to the prompt or press Q.

3 Key in the following: `A:\>`**SET U** Enter

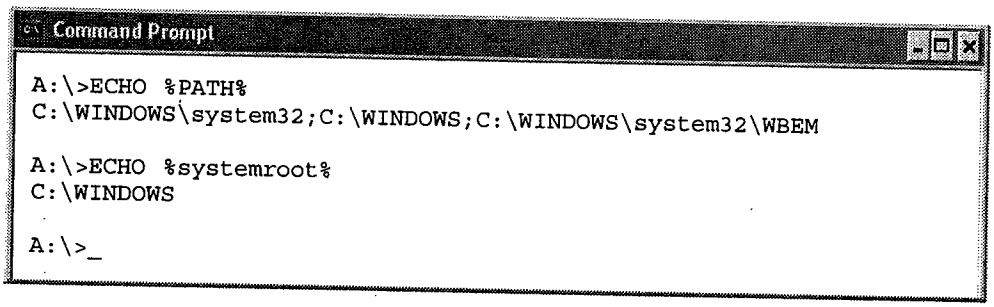
4 Key in the following: `A:\>`**SET S** Enter




**WHAT'S HAPPENING?** By using the SET command with a letter of the alphabet, all environmental variables that began with that letter were displayed. If you wanted to see the value of an environmental variable, you may do so with the ECHO command, provided that you enclose the environmental variable name you are seeking with percent signs.

5 Key in the following: `A:\>`**ECHO %PATH%** Enter

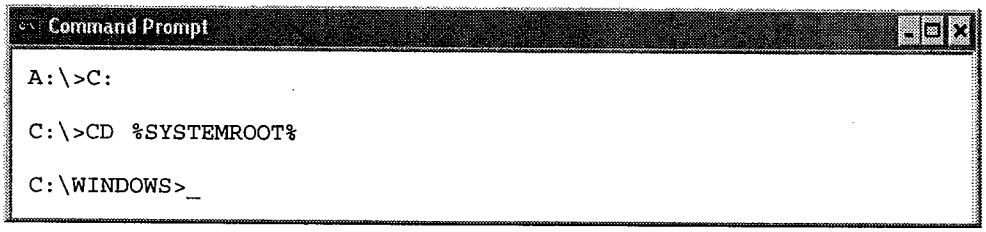
6 Key in the following: `A:\>`**ECHO %systemroot%** Enter



**WHAT'S HAPPENING?** By surrounding the environmental name with percent signs, you see the value for the variable you requested. As you can see, the case you use does not matter. You may also use the environmental variable with commands.

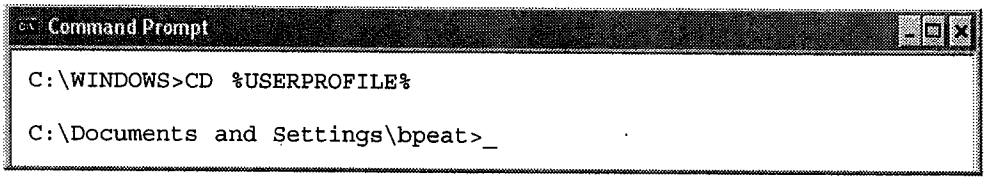
**7** Key in the following: `A:\>`**C:** Enter

**8** Key in the following: `C:\>`**CD %systemroot%** Enter



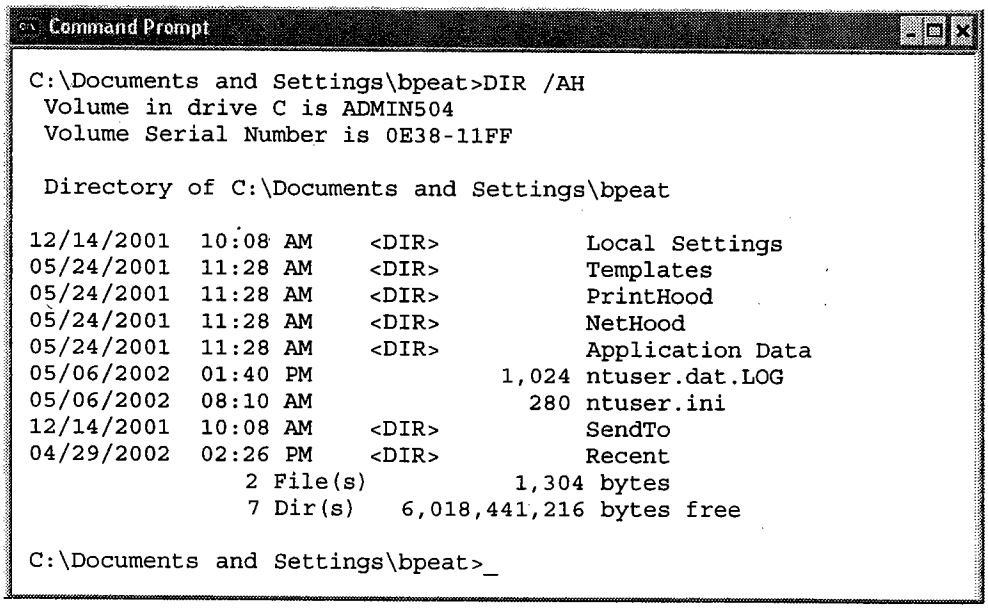

**WHAT'S HAPPENING?** Instead of keying in CD \WINDOWS, you used the environmental variable %SYSTEMROOT%, which changed your location to the value held by the environmental variable %SYSTEMROOT%, in this case, C:\WINDOWS.

**9** Key in the following: `C:\WINDOWS>`**CD %userprofile%** Enter



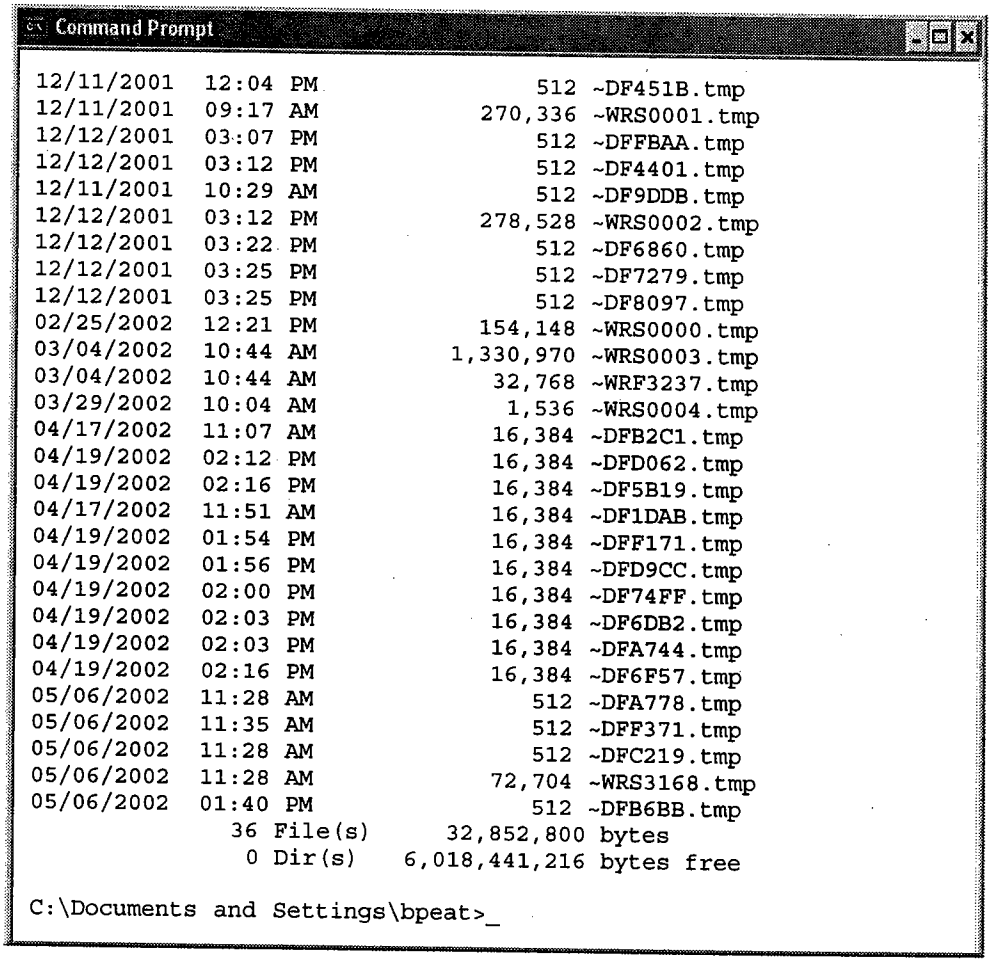

**WHAT'S HAPPENING?** Again, you used an environmental variable to change directories. The displayed prompt represents your personal user directory. Using the environmental variables can be a useful shortcut.

**10** Key in the following: `C:\Documents and Settings\bpeat>`**DIR /AH** Enter




**WHAT'S HAPPENING?** Local Settings is a hidden directory that contains settings that are specific for this user.

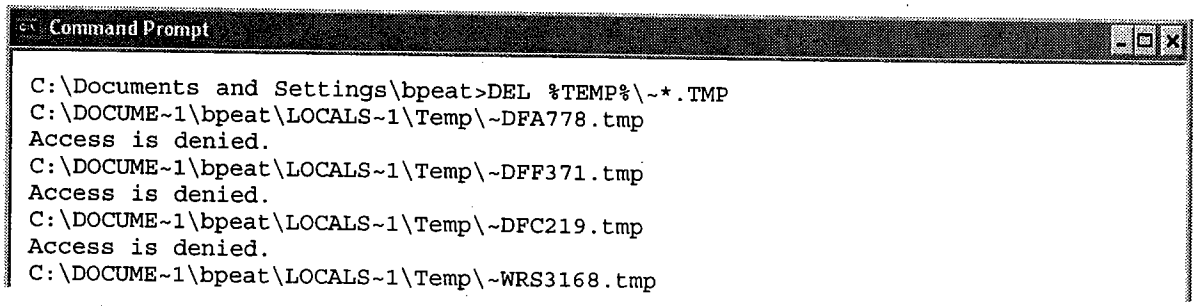
**11** Key in the following at your personal user prompt:
**DIR "Local Settings"\TEMP\~*.tmp** Enter




**WHAT'S HAPPENING?** You may have fewer files then those that are listed here (or no files). The TEMP directory is where Windows keeps temporary files that it is supposed to delete when you finish using a program. Often these files are not deleted. Rather than having to key in a long path name with the DEL command (DEL C:\Documents and Settings\bpeat\"LOCAL SETTINGS"\TEMP\~*.TMP), you can use the environmental variable name.

**12** Key in the following:
C:\Documents and Settings\bpeat>**DEL %TEMP%\~*.tmp** Enter

**13** Key in the following:
C:\Documents and Settings\bpeat>**DIR %TEMP%\~*.tmp** Enter



```
The process cannot access the file because it is being used by another process.
C:\DOCUME~1\bpeat\LOCALS~1\Temp\~DFB6BB.tmp
Access is denied.

C:\Documents and Settings\bpeat>DIR %TEMP%\~*.TMP
 Volume in drive C is ADMIN504
 Volume Serial Number is 0E38-11FF

 Directory of C:\DOCUME~1\bpeat\LOCALS~1\Temp

05/06/2002  11:28 AM               512 ~DFA778.tmp
05/06/2002  11:35 AM               512 ~DFF371.tmp
05/06/2002  11:28 AM               512 ~DFC219.tmp
05/06/2002  11:28 AM           134,144 ~WRS3168.tmp
05/06/2002  01:40 PM               512 ~DFB6BB.tmp
               5 File(s)        136,192 bytes
               0 Dir(s)   6,051,315,712 bytes free

C:\Documents and Settings\bpeat>_
```

**WHAT'S HAPPENING?** You may or may not get the message "Access Denied" or "The process cannot access the file because it is being used by another process." depending on what is currently in use on your system. You have quickly deleted the **~.TMP** files that are not currently in use by the operating system, using an environmental variable.

**14** Key in the following: `C:\Documents and Settings\bpeat>`**CD ** Enter

**15** Key in the following: `C:\>`**A:** Enter

**WHAT'S HAPPENING?** You have returned to the root of C, then returned to Drive A.

## 11.22 Using Set and the Environment in Batch Files

You have been using the built-in environmental variables that Windows sets and uses. You can also set your own environmental variables, giving them both a name and a value in a batch file as well as at the command line. Once you set the variable, you may use it in a batch file. However, any variables that are set are only good for that session of the Command Line window. Once you exit the command prompt, those values are no longer available the next time you open the command prompt.

## 11.23 Activity: Using SET and the Environment in Batch Files

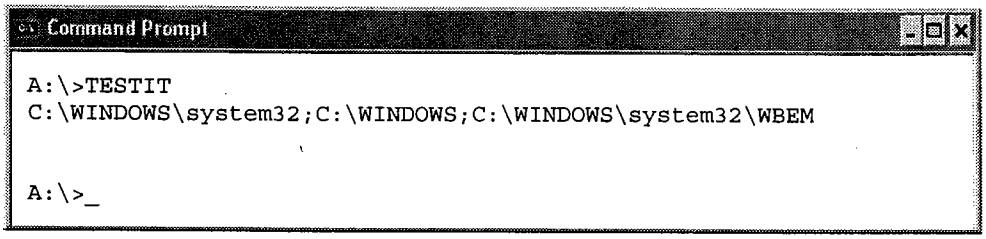
*Note:* The DATA disk should be in Drive A. The displayed prompt is A:\>.

**1** Close the Command Prompt window, and reopen it to begin a new DOS session. Return to the A:\> prompt.

**2** Write and save the following batch file called **TESTIT.BAT**:

```
@ECHO OFF
ECHO %PATH%
ECHO.
```

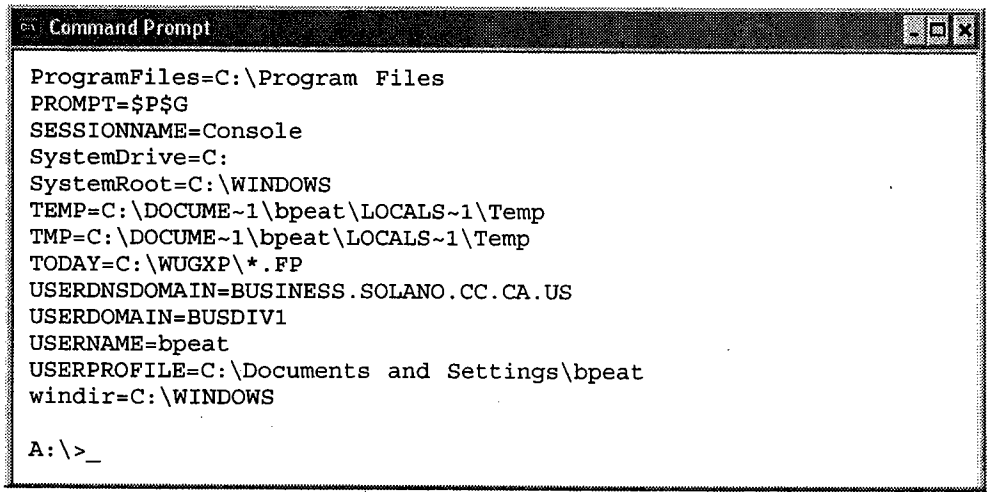
3 Key in the following: A:\>**TESTIT** Enter




**WHAT'S HAPPENING?** The screen display created by this batch file showed the path used in a Command Line window on your system. Notice that it did not return the word PATH but the value stored in the environmental variable "PATH." You can set an environmental value and then use it in a batch file.

4 Key in the following: A:\>**SET TODAY=C:\WUGXP*.FP** Enter

5 Key in the following: A:\>**SET** Enter

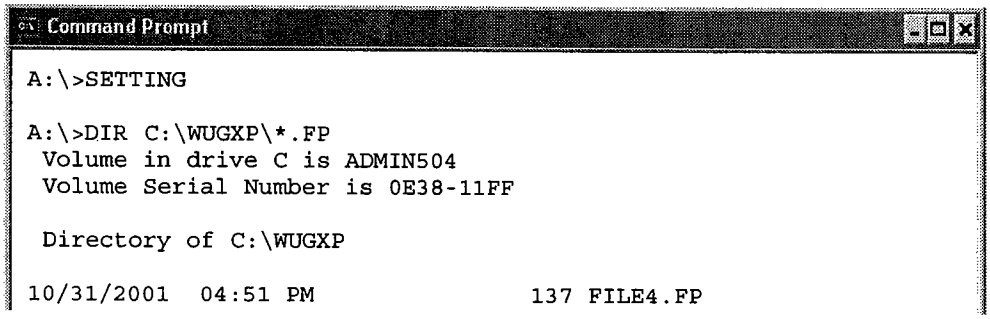



**WHAT'S HAPPENING?** You now have a value for TODAY, which you set in the environment as **C:\WUGXP*.FP**. Now, as long as you do not close the Command Prompt window, you can use it in a batch file. When you close the Command Prompt window, the environmental variables you set there will disappear.

6 Write and save the following batch file called **SETTING.BAT**:

```
DIR %today%
ECHO %TODAY%
```

7 Key in the following: A:\>**SETTING** Enter

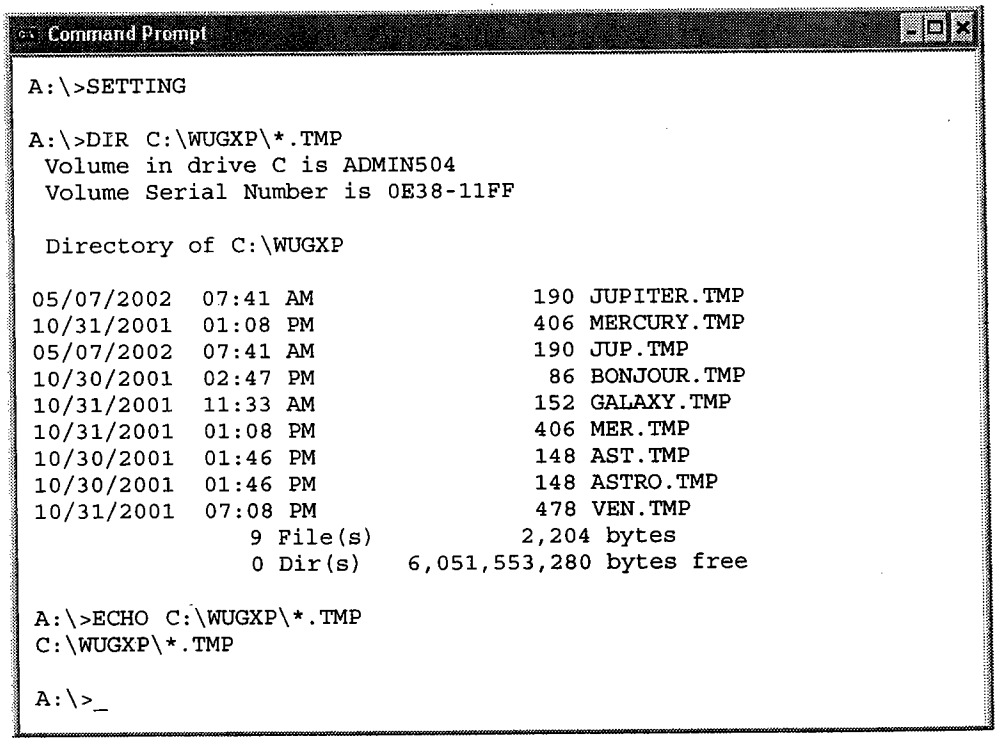


```
10/31/2001  04:51 PM               137 FILE2.FP
10/31/2001  04:51 PM               137 FILE3.FP
               3 File(s)            411 bytes
               0 Dir(s)   6,051,553,280 bytes free

A:\>ECHO C:\WUGXP\*.FP
C:\WUGXP\*.FP

A:\>_
```

**WHAT'S HAPPENING?** Your batch file needed a value for %today%. The percent signs indicate that the value was in the environment. It substituted **C:\WUGXP*.FP** for %today% and for %TODAY%. Case does not matter with environmental variables. You can use another value.

**8** Key in the following: `A:\>`**SET today=C:\WUGXP*.TMP** Enter

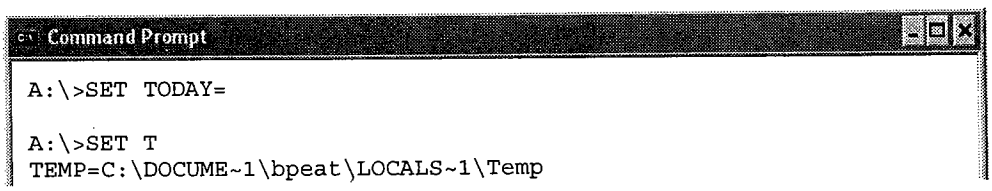
**9** Key in the following: `A:\>`**SETTING** Enter




**WHAT'S HAPPENING?** Since you changed the value of %TODAY% from **C:\WUGXP*.FP** to **C:\WUGXP*.TMP**, the batch file knew to get only the value in the environment called %TODAY%. To eliminate the value, you must set it to nothing.

**10** Key in the following: `A:\>`**SET TODAY=** Enter

**11** Key in the following: `A:\>`**SET T** Enter



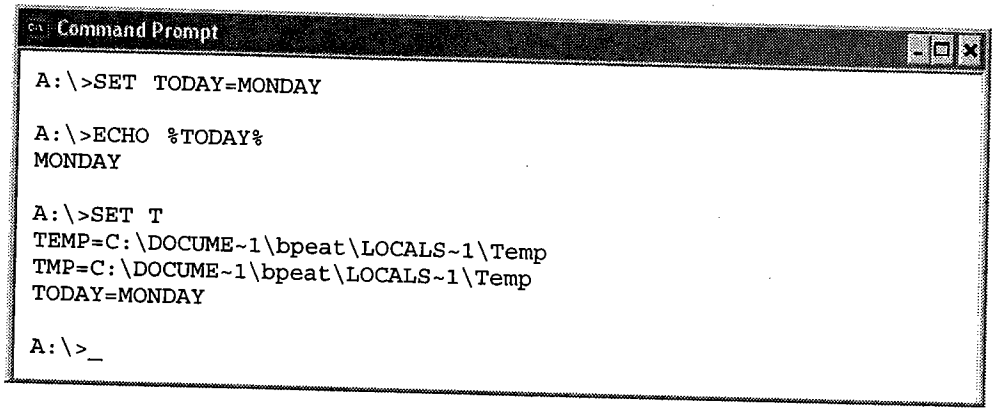
```
TMP=C:\DOCUME~1\bpeat\LOCALS~1\Temp

A:\>_
```

**WHAT'S HAPPENING?** You no longer have an environmental value called TODAY. That environmental variable would have been eliminated automatically if you had closed and reopened the Command Prompt window.

**12** Key in the following: `A:\>`**SET TODAY=MONDAY** Enter

**13** Key in the following: `A:\>`**ECHO %TODAY%** Enter

**14** Key in the following: `A:\>`**SET T** Enter

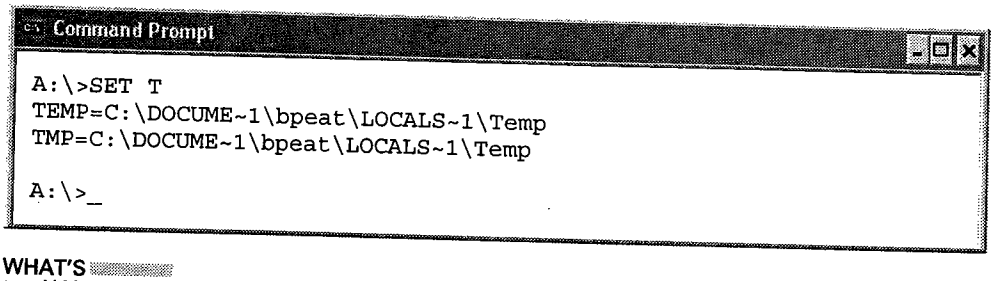



**WHAT'S HAPPENING?** You have set a new environmental variable with the value of MONDAY. You have used the variable syntax %VARIABLENAME% to display the value of the variable. You have also used the SET T command to see any current environment variables that begin with T.

**15** Close the Command Prompt window.

**16** Reopen the Command Prompt window, and return to the A prompt.

**17** Key in the following: `A:\>`**SET T** Enter




**WHAT'S HAPPENING?** The TODAY variable is no longer there. You can create a useful batch file that you can use during a DOS session. You do not often want to add a directory to your PATH statement, but perhaps you will be doing a lot of work at the Command Prompt using files that are in the root of the A drive. To do this by hand would involve keying in the entire path you currently have and adding your new directory to the end. There is an easier way to do it using the environment.

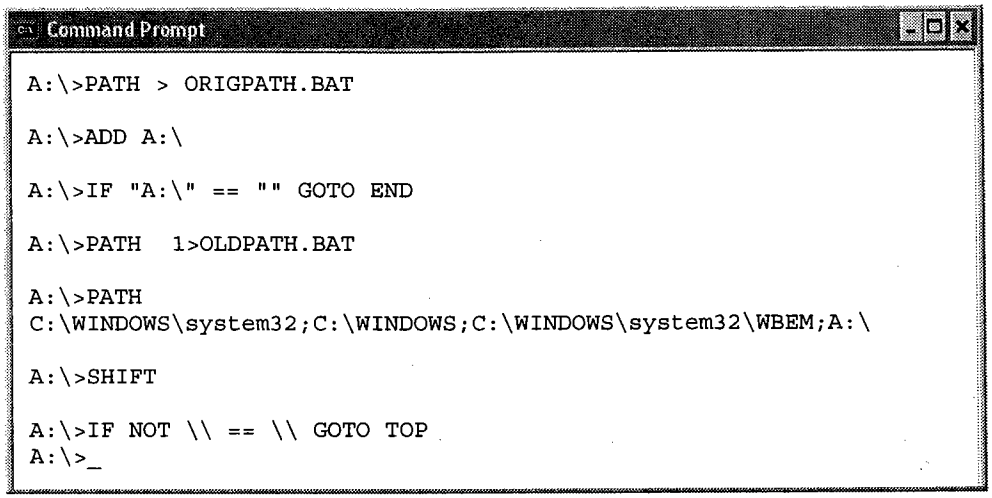
*Note:* The default prompt is A:\>.

**18** Write and save the following batch file called **ADD.BAT**:

```
IF "%1"=="" GOTO END
PATH > OLDPATH.BAT
:TOP
PATH %PATH%;%1
SHIFT
IF NOT \%1\==\\ GOTO TOP
:END
```

**19** Key in the following: `A:\>`**PATH > ORIGPATH.BAT** Enter

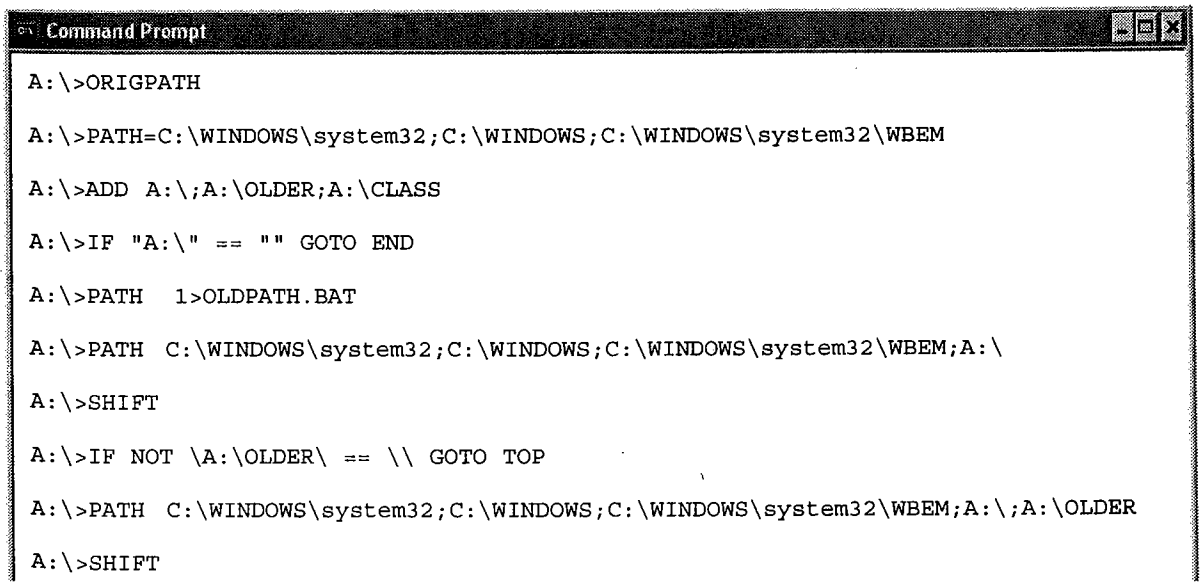
**20** Key in the following: `A:\>`**ADD A:** Enter




**WHAT'S HAPPENING?** To preserve your default path, you saved it to a file called **ORIGPATH.BAT**. You then used your new batch file, **ADD.BAT**, and added the **A:** root directory to the path. You can add more than one directory.

**21** Key in the following: `A:\>`**ORIGPATH** Enter

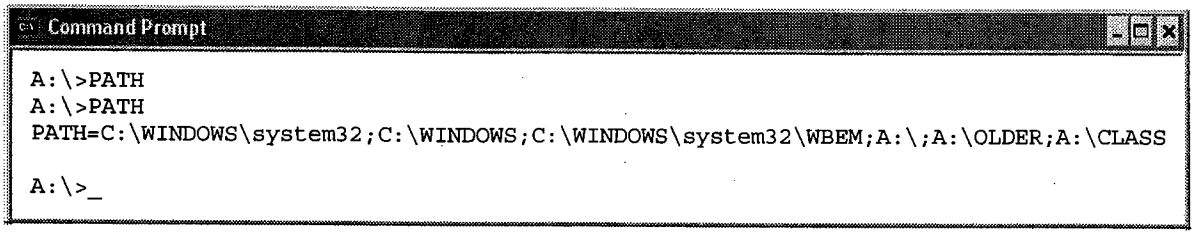
**22** Key in the following: `A:\>`**ADD A:\;A:\OLDER;A:\CLASS** Enter



```
A:\>IF NOT \A:\CLASS\ == \\ GOTO TOP

A:\>PATH
C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\system32\WBEM;A:\;A:\OLDER;A:\CLASS

A:\>SHIFT

A:\>IF NOT \\ == \\ GOTO TOP
A:\>_
```

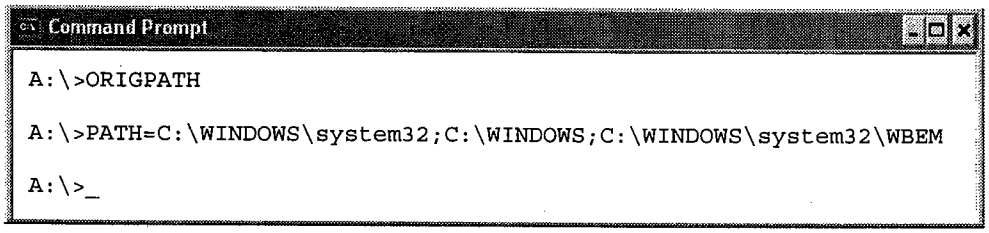
**WHAT'S HAPPENING?** You have quickly added new directories to your path.

**23** Key in the following: `A:\>`**PATH** Enter




**WHAT'S HAPPENING?** You keyed in **PATH** to confirm that you added subdirectories. To return to your original path, you created **ORIGPATH.BAT**.

**24** Key in the following: `A:\>`**ORIGPATH** Enter




**WHAT'S HAPPENING?** You have returned to the original path.

## 11.24 The DIRCMD Environmental Variable

As has been discussed, the environment is an area that is set aside in memory. In addition to being able to place and use variables in the environment, you can preset DIR command parameters and switches by including the SET command with the DIRCMD environmental variable. Keying in SET by itself will tell you what is in the environment. You can use the DIRCMD variable and ERRORLEVEL to write a batch file that will allow you to change the way DIR displays information for the current command prompt work session.

## 11.25 Activity: Using DIRCMD

*Note:* The DATA disk should be in Drive A. The displayed prompt is A:\>.

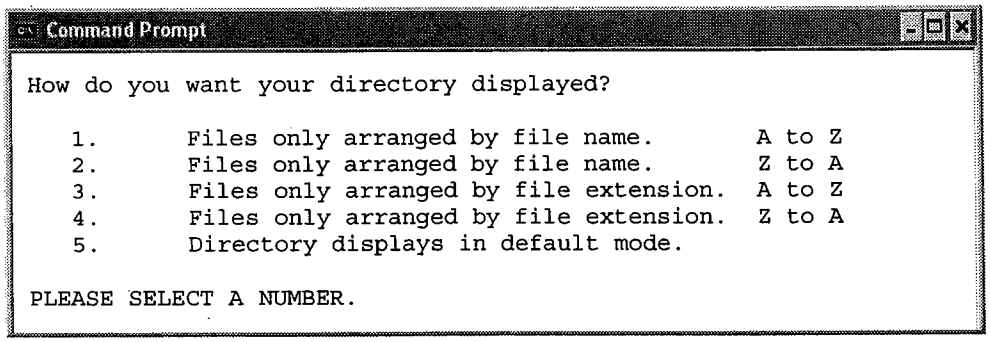
**1** Create the following batch file called **MY.BAT**:

```
@ECHO OFF
CLS
```

```
ECHO.
ECHO.
ECHO How do you want your directory displayed?
ECHO.
ECHO  1.  Files only arranged by file name.      A to Z
ECHO  2.  Files only arranged by file name.      Z to A
ECHO  3.  Files only arranged by file extension. A to Z
ECHO  4.  Files only arranged by file extension. Z to A
ECHO  5.  Directory displays in default mode.
ECHO.
ECHO PLEASE SELECT A NUMBER.
ECHO.
REPLY
ECHO.
IF ERRORLEVEL 49 IF NOT ERRORLEVEL 50 SET DIRCMD=/ON /A-D
IF ERRORLEVEL 50 IF NOT ERRORLEVEL 51 SET DIRCMD=/O-N /A-D
IF ERRORLEVEL 51 IF NOT ERRORLEVEL 52 SET DIRCMD=/OE /A-D
IF ERRORLEVEL 52 IF NOT ERRORLEVEL 53 SET DIRCMD=/O-E /A-D
IF ERRORLEVEL 53 IF NOT ERRORLEVEL 54 SET DIRCMD=
```

**WHAT'S HAPPENING?** You have created a batch file to set the DIRCMD environmental variable.

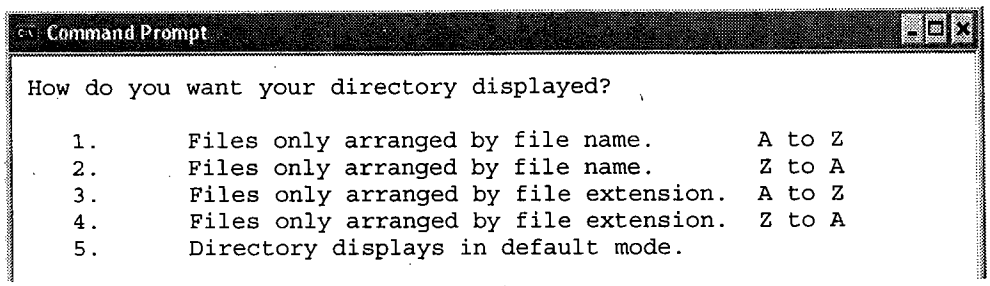
2 Key in the following: `A:\>`**MY** Enter




**WHAT'S HAPPENING?** The batch file is asking you to select how you want your batch files displayed. You want your files displayed in file extension order in descending order (Z–A).

3 Key in the following: **4**

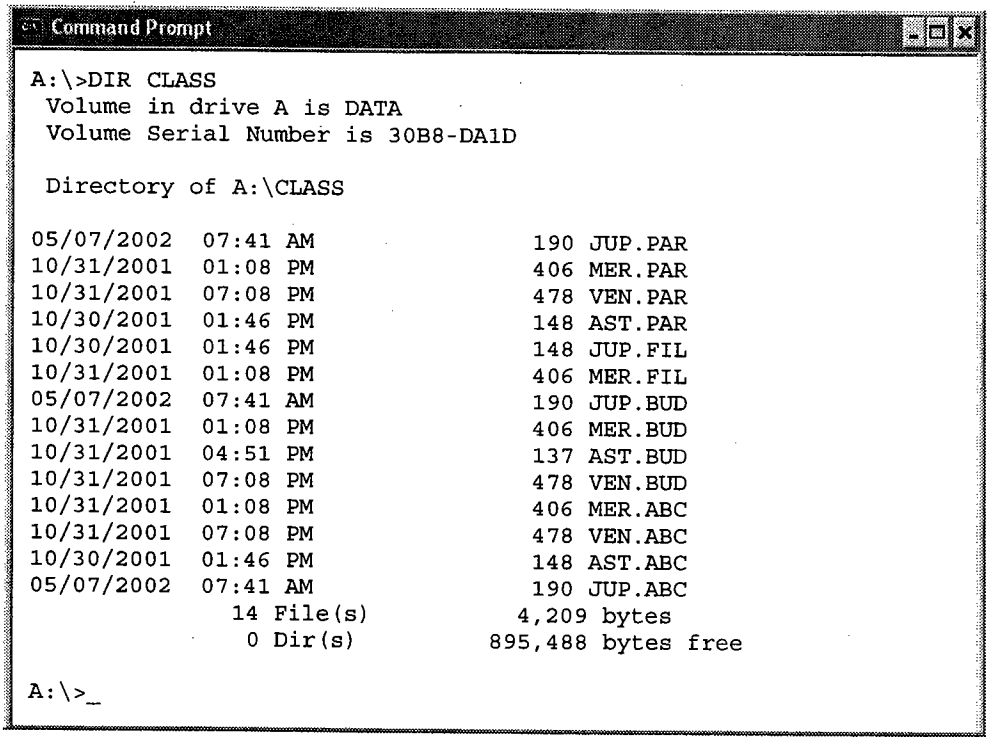
4 Key in the following: `A:\>`**SET D** Enter



```
PLEASE SELECT A NUMBER.



A:\>SET D
DIRCMD=/O-E /A-D

A:\>_
```

**WHAT'S HAPPENING?** The 4 that you keyed in disappeared, but you can see that you did indeed set an environmental variable. Now, during this Command Line session, whenever you key in **DIR**, it will automatically arrange the files by extension in reverse order. You can see that you have established the environmental variable for DIRCMD to equal the switches /O-E and /A-D.

**5** Key in the following: `A:\>`**DIR CLASS** Enter

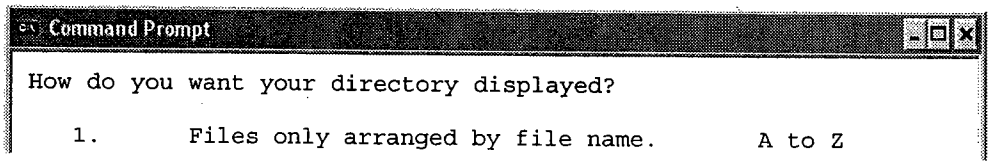



**WHAT'S HAPPENING?** The files are arranged by file extension in reverse alphabetical order. Until you change the values, or close this Command Prompt session, every time you issue the DIR command it will display file names in reverse alphabetical order by file extension.

**6** Key in the following: `A:\>`**MY** Enter

**7** Press **5**

**8** Key in the following: `A:\>`**SET D** Enter



```
   2.      Files only arranged by file name.       Z to A
   3.      Files only arranged by file extension.  A to Z
   4.      Files only arranged by file extension.  Z to A
   5.      Directory displays in default mode.

PLEASE SELECT A NUMBER.



A:\>SET D
Environment variable D not defined

A:\>_
```

**WHAT'S HAPPENING?** You returned the default DIRCMD environmental variable to its default value. DIRCMD is no longer defined.

## 11.26 The FOR...IN...DO Command

The FOR...IN...DO command can be issued at the command line or placed in a batch file. This command allows repetitive processing. FOR allows you to use a single command to issue several commands at once. The command can DO something FOR every value IN a specified set. The basic syntax at the command line is:

```
FOR %variable IN (set) DO command [command-parameters]

 %variable           Specifies a replaceable parameter.
 (set)               Specifies a set of one or more files. Wildcards
                     may be used.
 command             Specifies the command to carry out for each
                     file.
 command-parameters  Specifies parameters or switches for the
                     specified command.

To use the FOR command in a batch program, specify %%variable
instead of %variable.
```

The FOR command was greatly expanded with the release of Windows 2000 Professional. For full details, see Appendix H or key in FOR /? at the prompt.

The batch file variable is an arbitrary single letter. The double percent sign with a letter (%%a) distinguishes the batch file variable from the replaceable parameter (%1). The difference between a variable and a parameter is not complicated. The FOR statement tells the operating system to get a value from the set you have chosen. After it executes the command that appears after DO, the FOR command looks for the next value in the set. If it finds another value, %%a will represent something new, and the command will be executed with the new value. If there are no more values in the set, the FOR command stops processing.

If you consider the GOTO label as a *vertical* loop, you can consider the FOR...IN...DO as a *horizontal* loop. You do not need to use the letter a. You may use any letter—a, c, x, etc. The parameter value, on the other hand, is set before the batch file begins processing. Remember, the operating system gets the value from the position in the command line. The set is always enclosed in parentheses. The values in the set, either data or file names, will be used to DO some command. The

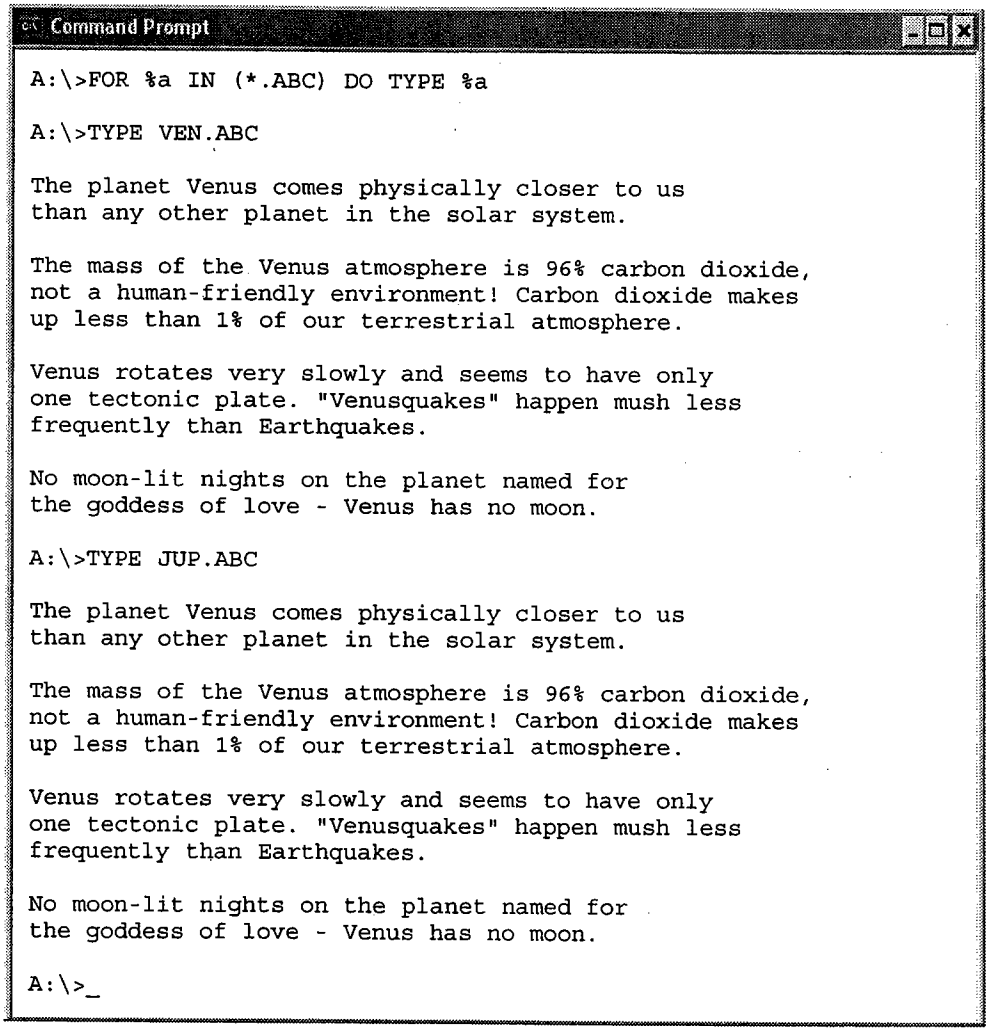
items in the set must be separated by spaces or commas. You may also use wildcards in a set.

## 11.27 Activity: Using the FOR...IN...DO Command

*Note 1:* The DATA disk should be in Drive A. The displayed prompt is A:\>.

*Note 2:* Look at the command line you are going to use in step 1. In English, the command says: Using the variable %a to hold each value in the set (what is in parentheses), do the command (TYPE) to each value in the set (%a).

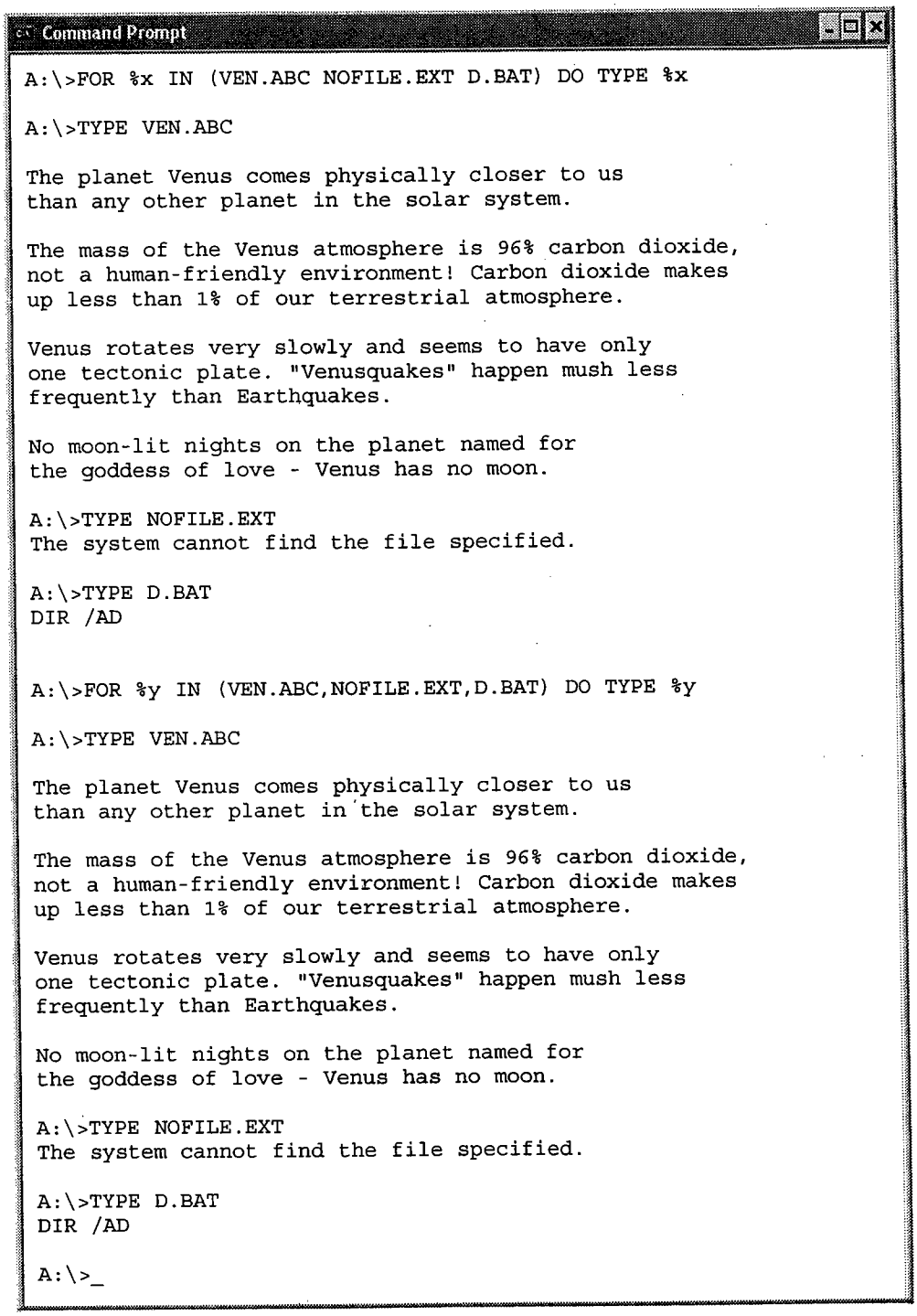
**1** Key in the following: `A:\>`**FOR %a IN (*.ABC) DO TYPE %a** Enter




**WHAT'S HAPPENING?** FOR...IN...DO processed every item in the set as indicated below. Besides using wildcards, you can also be specific.

**2** Key in the following:
`A:\>`**FOR %x IN (VEN.ABC NOFILE.EXT D.BAT) DO TYPE %x** Enter

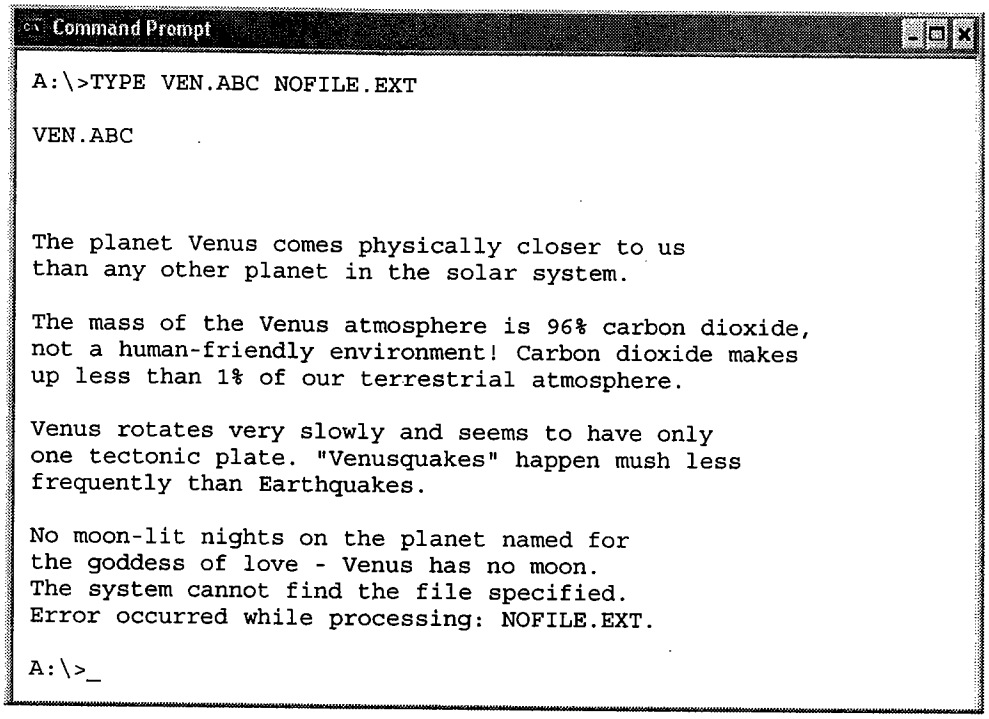
**3** Key in the following:
`A:\>`**FOR %y IN (VEN.ABC,NOFILE.EXT,D.BAT) DO TYPE %y** Enter




**WHAT'S HAPPENING?** There are some important things to notice about these command lines. First, both a space and a comma between items in a set work the same way. Second, the variable letter you choose is not important. In the first case, x was chosen—in the second, y. This command line is case sensitive. If you had keyed in **FOR %b IN (VEN.ABC NOFILE.EXT D.BAT) DO TYPE %B**, the difference between b and B would have made the command line invalid. Even when there was an invalid file (**NOFILE.EXT**), the command line continued processing the other file names in the command. You did not need to worry about testing for null values.

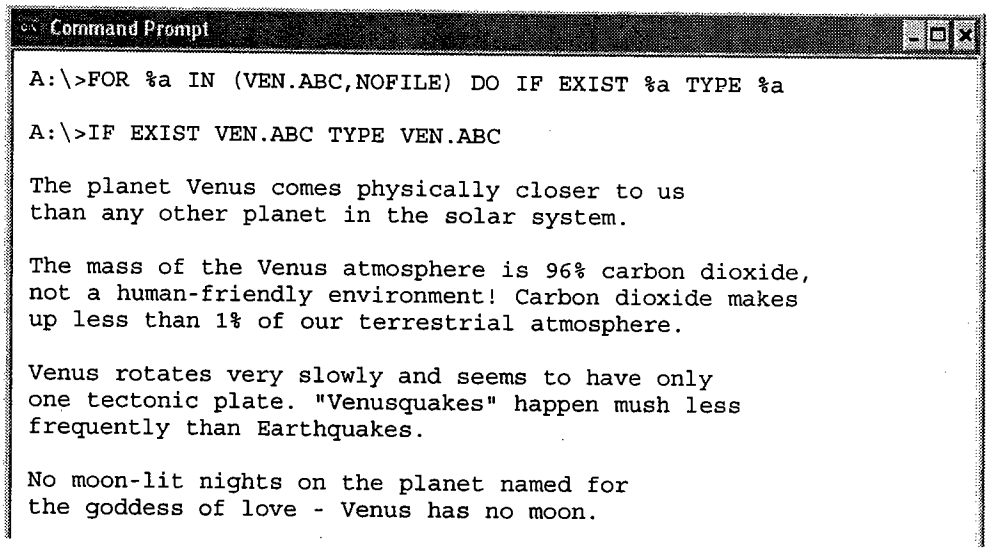
This command works the same when placed in a batch file, only you must use %%. However, it appears that this works no differently than had you keyed in TYPE VEN.ABC NOFILE.EXT.

**4** Key in the following: A:\>**TYPE VEN.ABC NOFILE.EXT** Enter




**WHAT'S HAPPENING?** You see an error message since NOFILE.EXT does not exist. You can use a test to test for an existence of a file in conjunction with the FOR...IN...DO so that the TYPE command will only display the files it finds and will not display any error messages.

**5** Key in the following: A:\>
**FOR %a IN (VEN.ABC,NOFILE) DO IF EXIST %a TYPE %a** Enter

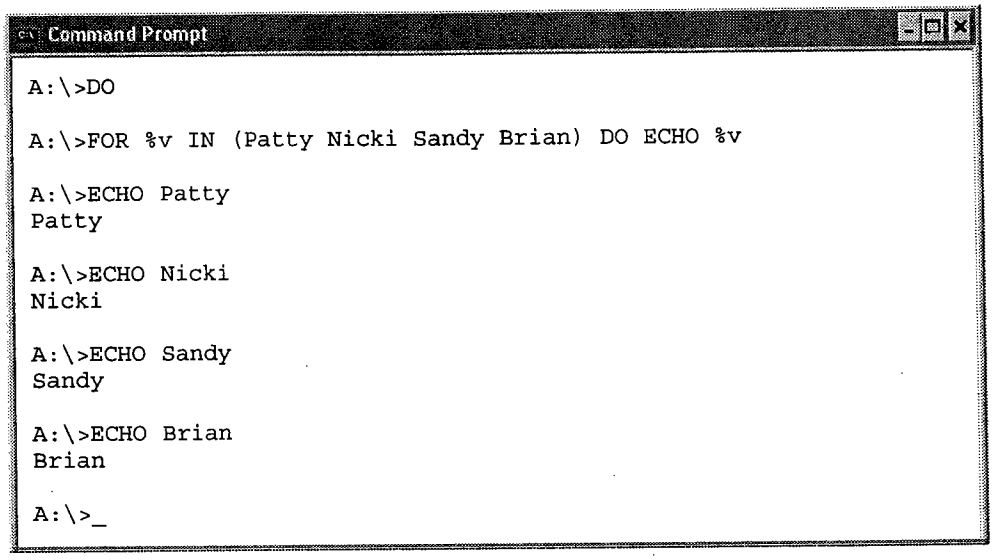


```
A:\>IF EXIST NOFILE TYPE NOFILE

A:\>_
```

**WHAT'S HAPPENING?** Now that you checked to see if there is a file, you no longer see the error message. You can also test for character strings supplied in the set.

6 Create and save the following batch file called **DO.BAT** and key in the following:
**FOR %%v IN (Patty Nicki Sandy Brian) DO ECHO %%v**

7 Key in the following: A:\>**DO** Enter




**WHAT'S HAPPENING?** As you can see, the ECHO command was carried out for each item in the set. It substituted each value (**Patty, Nicki, Sandy,** and **Brian**) for the replaceable parameter in the ECHO command. In this example, spaces were used to separate the values, but you could have also used commas. The advantage to using this command is that you do not have to write it with a command on each line, as in:

**ECHO Patty**
**ECHO Nicki**
**ECHO Sandy**
**ECHO Brian**

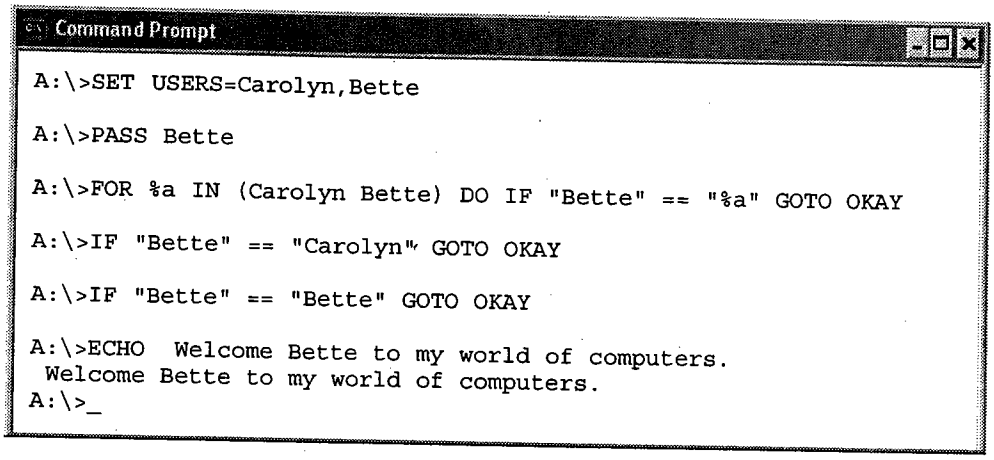
Another advantage of the FOR...IN...DO method is that you can set values in the environment and then use them in a batch file.

8 Write and save a batch file called **PASS.BAT** and key in the following:
**FOR %%a IN (%USERS%) DO IF "%1"=="%%a" GOTO OKAY** Enter
**:NO** Enter
**ECHO You, %1, are NOT allowed in the system.** Enter
**GOTO END** Enter
**:OKAY** Enter
**ECHO Welcome, %1, to my world of computers.** Enter
**:END**

**WHAT'S HAPPENING?** You have combined several features in this FOR..IN..DO statement. You have used an environmental variable in the set (**%USERS%**). The percent signs surrounding the value tell the FOR command to use the environmental variable called USERS. You have also used an IF statement. If what the user keys in is in the environment, then it is a true statement and the batch file will go to the :OKAY label. If what the user keys in is false and not equal to the environmental variable, then the batch file falls through to the next line. First, you need to set the environmental variable. (Use upper and lower case exactly as shown.)

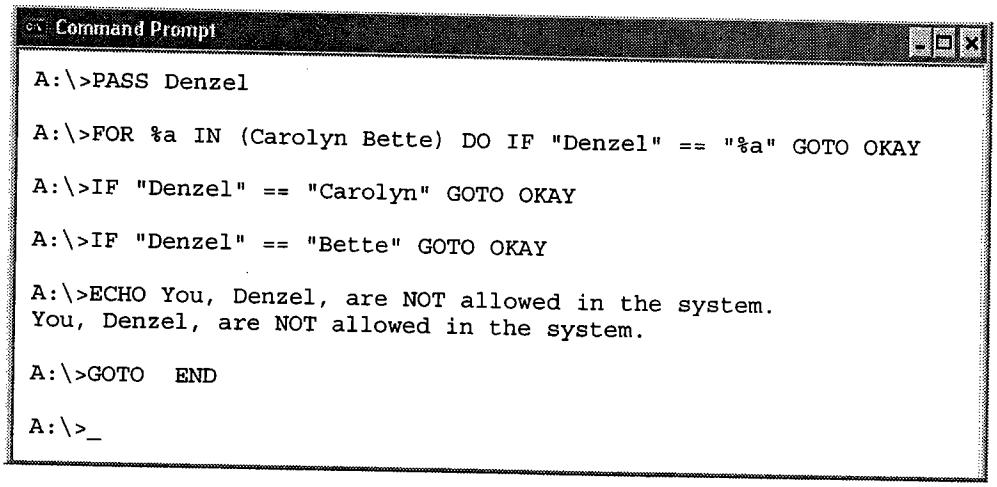
**9** Key in the following: `A:\>`**SET USERS=Carolyn,Bette** Enter

**10** Key in the following: `A:\>`**PASS Bette** Enter




**WHAT'S HAPPENING?** You set the environmental values for USERS. You then executed the **PASS.BAT** batch file. It worked as directed because the statement was true. What if it were false?

**11** Key in the following: `A:\>`**PASS Denzel** Enter



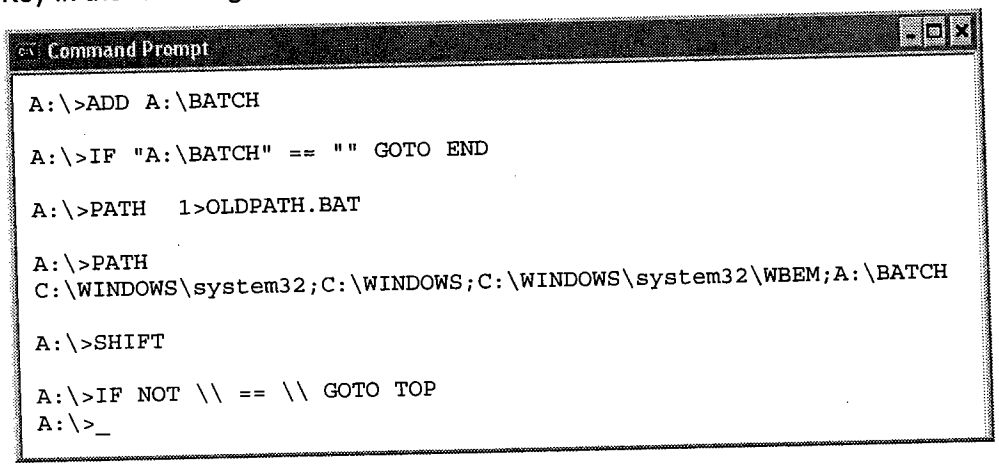

**WHAT'S HAPPENING?** The statement was false and the batch file behaved accordingly. FOR..IN..DO can also be used with replaceable parameters, file names, and wildcards. You are going to take another look at **UPDATE.BAT**.

**12** Use any text editor and edit and save the **UPDATE.BAT** file so it looks as follows. Be sure you include two percent signs preceding "v."

```
:DOIT
IF "%1"=="" GOTO END
FOR %%v IN (%1) DO COPY %%v /b + > NUL
SHIFT
PAUSE
GOTO DOIT
:END
```

**WHAT'S HAPPENING?** You now can process any number of parameters that appear in the command line. There can be a problem with this batch file. As written, this batch file will copy the newly updated files to the current default directory. Thus, it is a good idea, in general, to place all your batch files in a subdirectory called **BATCH** and set your path to include the **BATCH** directory. If you do not have the BATCH directory on your root of your DATA disk, create it now.

**13** Key in the following: `A:\>`**ADD A:\BATCH** Enter

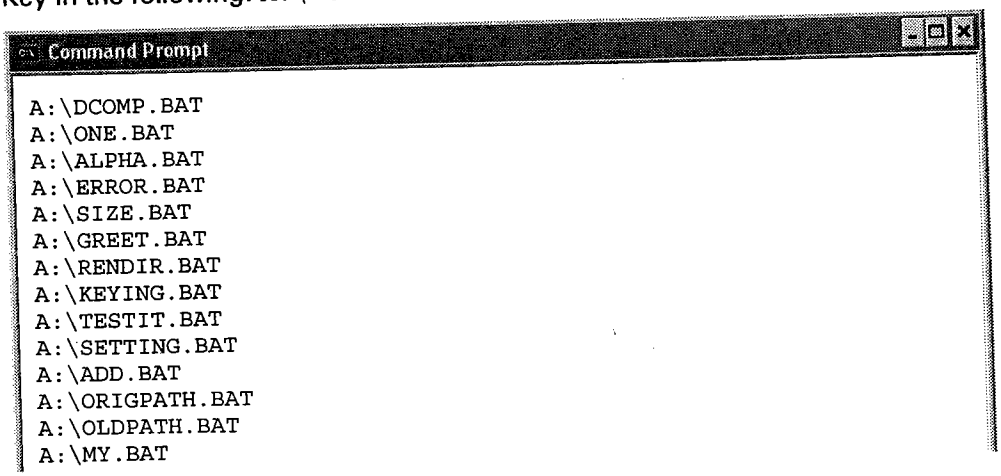



**WHAT'S HAPPENING?** You have added the **BATCH** subdirectory to your current path. Now you will move all the batch files, as well as the **REPLY** files, into the **BATCH** subdirectory.

**14** Key in the following: `A:\>`**MOVE *.BAT BATCH** Enter

**15** Key in the following: `A:\>`**MOVE REPLY.* BATCH** Enter

**16** Key in the following: `A:\>`**COPY BATCH\ADD.BAT** Enter



```
A:\DO.BAT
A:\PASS.BAT
A:\UPDATE.BAT
A:\b.bat
A:\TEST.BAT
A:\EXAMPLE.BAT
A:\D.BAT
A:\S.BAT
A:\BOG.BAT
A:\N.BAT
A:\log.bat

A:\>MOVE REPLY.* BATCH
A:\REPLY.SCR
A:\REPLY.COM

A:\>COPY BATCH\ADD.BAT
        1 file(s) copied.

A:\>_
```

**WHAT'S HAPPENING?** You have moved all your batch files to the **BATCH** subdirectory and you included the **REPLY** files, because batch files use these programs. This grouping allowed you to clean up the root directory of the DATA disk. To ensure that you can use your batch files, you first added the **BATCH** subdirectory to the PATH statement. Then you copied the ADD.BAT file to the root.

**CAUTION: If you close the Command Prompt window, you will have to issue the following command to include the A:\BATCH directory in your path:**
**A:\ADD A:\BATCH**

**17** Key in the following: `A:\>`**DIR *.SWT** Enter

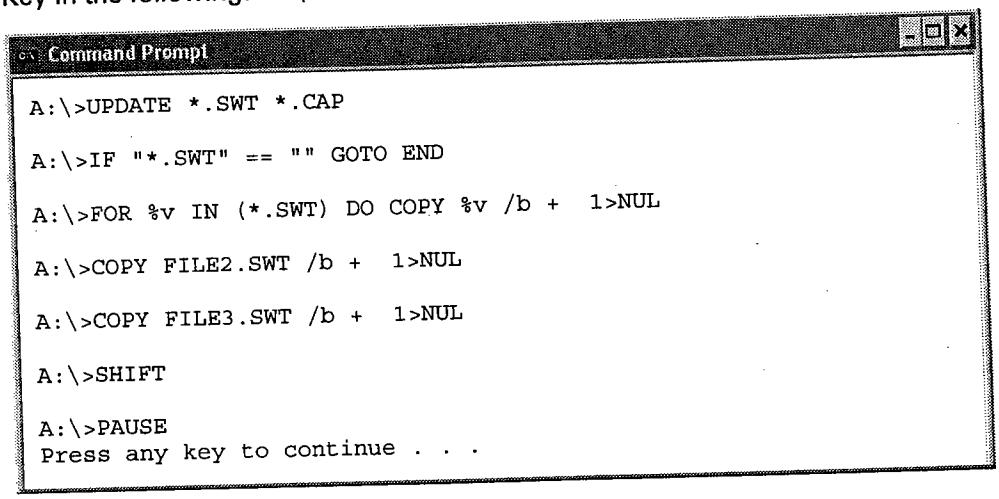
**18** Key in the following: `A:\>`**DIR *.CAP** Enter

Command Prompt

```
A:\>DIR *.SWT
 Volume in drive A is DATA
 Volume Serial Number is 30B8-DA1D

 Directory of A:\

10/31/2001  04:50 PM               138 FILE2.SWT
10/31/2001  04:50 PM               138 FILE3.SWT
               2 File(s)            276 bytes
               0 Dir(s)         893,440 bytes free

A:\>DIR *.CAP
 Volume in drive A is DATA
 Volume Serial Number is 30B8-DA1D

 Directory of A:\

07/31/2000  04:32 PM               260 STATE.CAP
04/23/2002  01:01 PM               260 SORTED.CAP
04/23/2002  01:04 PM               260 BYCITY.CAP
               3 File(s)            780 bytes
               0 Dir(s)         893,440 bytes free

A:\>_
```

**WHAT'S HAPPENING?** You can see the dates on these files. Now you are going to update them to the current date.

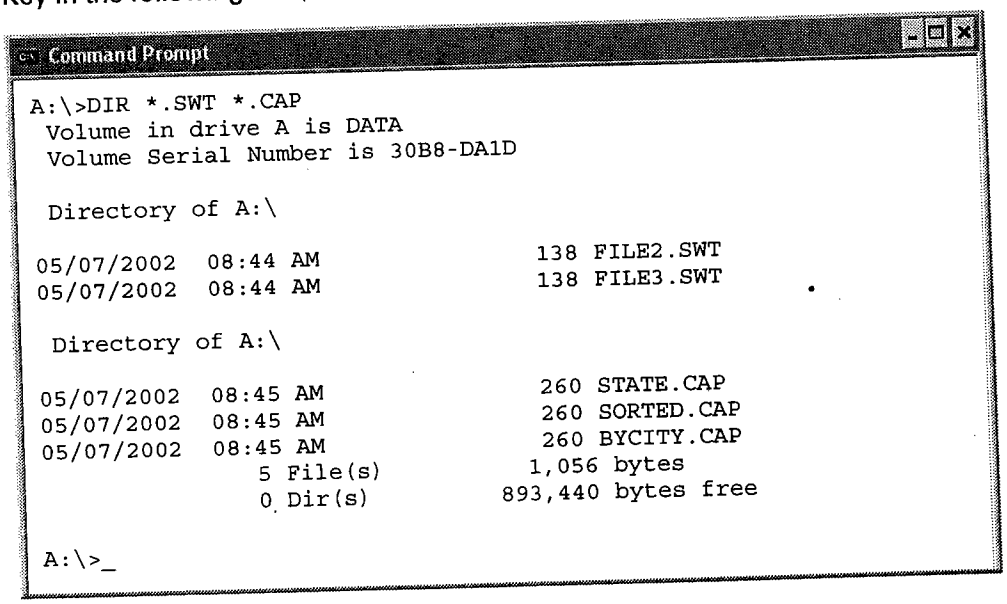
**19** Key in the following: `A:\>`**UPDATE *.SWT *.CAP** Enter




**WHAT'S HAPPENING?** You can see how FOR processed each item in the set, which it got from the command line (%1 was ***.SWT**). Each file that had a **.SWT** file extension was updated. Now the batch file is going to SHIFT and process the new item in the set (***.CAP**).

**20** Keep pressing Enter until you are back at the command prompt.

**21** Key in the following: `A:\>`**DIR *.SWT *.CAP** Enter



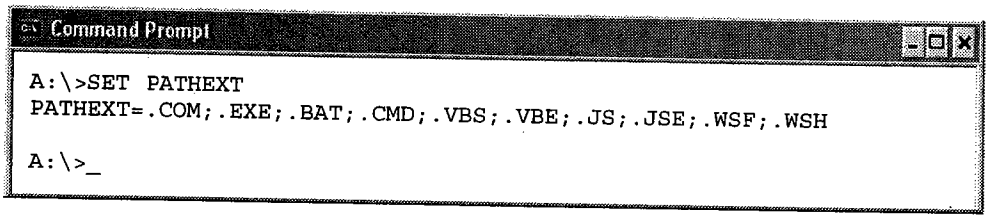

**WHAT'S HAPPENING?** Your dates will be different, but you have successfully changed the dates of the files using the FOR.. IN..DO command.

# 11.28 More Features of the FOR...IN...DO Command

Some of the features in of the FOR...IN...DO command are that you may list environmental variables so that they are divided and appear on separate lines, you may use the special tilde operator (~) to perform such tasks as stripping a file name of quotation marks and to expand a variable, and you may also use the /R parameter. The /R parameter is a recursive parameter. *Recursive* means that the command will search and perform actions on all subdirectories beneath it. You may also select specific text from ASCII files.

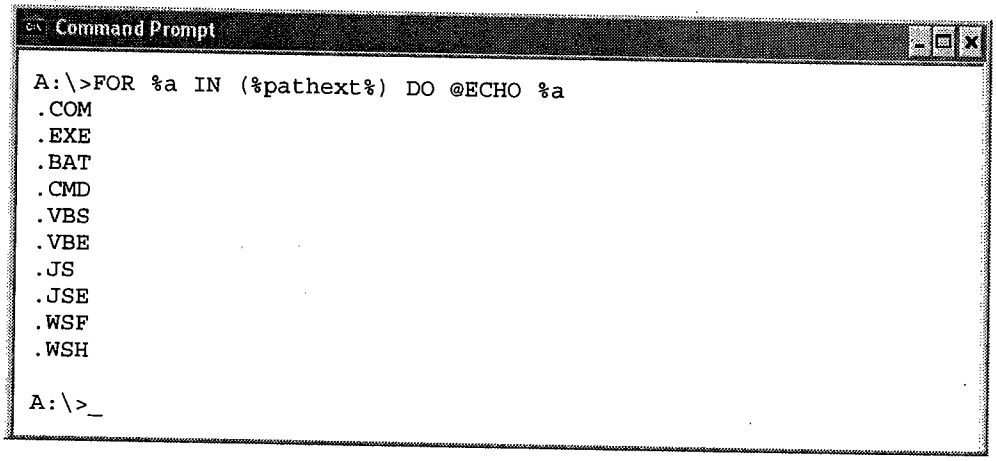
# 11.29 Activity: Using the Additional Features of the FOR...IN...DO Command

1 Key in the following: `A:\>`**SET PATHEXT** Enter




**WHAT'S HAPPENING?** The above list is a list of the file extensions that Windows XP Professional has associated with the application programs on your system. Your list may be different, depending on the software installed on your system. This list is separated by semicolons and appears on one line, sometimes making it difficult to read. You can use FOR...IN...DO to display the list one line at a time.

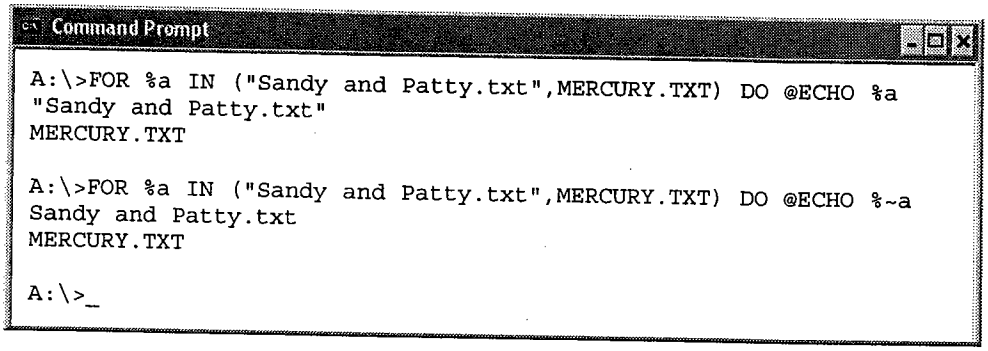
2 Key in the following:
`A:\>`**FOR %a IN (%pathext%) DO @ECHO %a** Enter




**WHAT'S HAPPENING?** Now your extensions are listed one line at a time. Files with spaces in their names provide certain challenges at the command line as well as in batch files.

3 Key in the following: A:\>**FOR %a IN ("Sandy and Patty.txt", MERCURY.TXT) DO @ECHO %a** Enter

4 Key in the following: A:\>**FOR %a IN ("Sandy and Patty.txt", MERCURY.TXT) DO @ECHO %~a** Enter

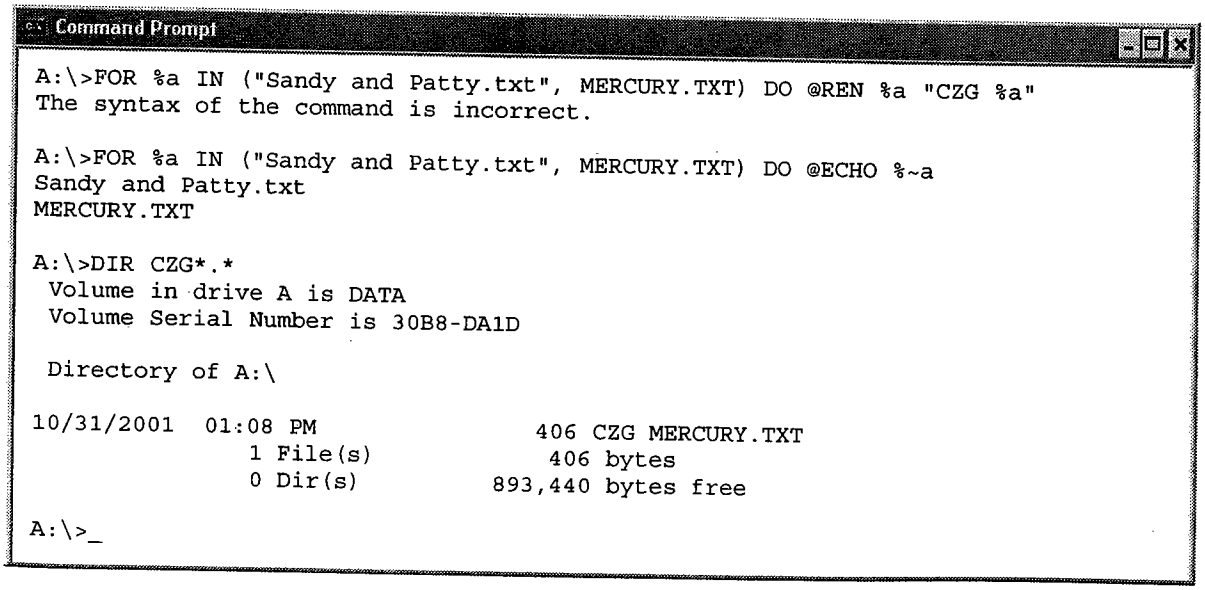



**WHAT'S HAPPENING?** In the second command, you added a tilde prior to the variable, a. This stripped the file name of its quotation marks. When you want to add a prefix or suffix to a long file name, you need to use quotation marks. However, if, in the set, you use quotation marks, when you try to rename a file, you will end up with extra quotation marks.

5 Key in the following: A:\>**FOR %a IN ("Sandy and Patty.txt", MERCURY.TXT) DO @REN %a "CZG %a"** Enter

6 Key in the following: A:\>**FOR %a IN ("Sandy and Patty.txt", MERCURY.TXT) DO @ECHO %~a** Enter

7 Key in the following: A:\>**DIR CZG*.*** Enter




**WHAT'S HAPPENING?** The file name with quotation marks did not get renamed because the syntax was incorrect. You had too many quotation marks.

8 Key in the following: A:\>**DEL CZG*.*** Enter

9 Key in the following (do not press Enter until you see Enter):
A:\>**FOR %a IN ("Sandy and Patty.txt", JUPITER.TXT) DO REN %a "Brian %~a"** Enter

```
Command Prompt
A:\>DEL CZG*.*

A:\>FOR %a IN  ("Sandy and Patty.txt", JUPITER.TXT) DO REN %a "Brian %~a"

A:\>REN "Sandy and Patty.txt" "Brian Sandy and Patty.txt"

A:\>REN JUPITER.TXT "Brian JUPITER.TXT"

A:\>_
```

WHAT'S HAPPENING? Because you stripped out the quotation marks, you could successfully rename your files. In step 10, overwrite the files, if necessary.

10 Key in the following: A:\>**COPY C:\WUGXP*.TXT TRIP** Enter

11 Key in the following:
A:\>**FOR /R %a IN (*SANDY*.*) DO @ECHO %a** Enter

12 Key in the following:
A:\>**FOR /R %a IN (*SANDY*.*) DO @ECHO %~nxa** Enter

```
Command Prompt
A:\>COPY C:\WUGXP\*.TXT TRIP
C:\WUGXP\ASTRO.TXT
C:\WUGXP\BORN.TXT
C:\WUGXP\DANCES.TXT
C:\WUGXP\HELLO.TXT
C:\WUGXP\Sandy and Patty.txt
C:\WUGXP\TITAN.TXT
C:\WUGXP\Sandy and Nicki.txt
C:\WUGXP\LONGFILENAME.TXT
C:\WUGXP\JUPITER.TXT
C:\WUGXP\LONGFILENAMED.TXT
C:\WUGXP\GALAXY.TXT
C:\WUGXP\LONGFILENAMING.TXT
C:\WUGXP\MERCURY.TXT
C:\WUGXP\PLANETS.TXT
C:\WUGXP\VENUS.TXT
       15 file(s) copied.

A:\>FOR /R %a IN (*SANDY*.*) DO @ECHO %a
A:\Brian Sandy and Patty.txt
A:\Sandy and Nicki.txt
A:\TRIP\Sandy and Patty.txt
A:\TRIP\Sandy and Nicki.txt
A:\OLDER\Sandy and Patty.txt
A:\OLDER\Sandy and Nicki.txt

A:\>FOR /R %a IN (*SANDY*.*) DO @ECHO %~nxa
Brian Sandy and Patty.txt
Sandy and Nicki.txt
Sandy and Patty.txt
Sandy and Nicki.txt
Sandy and Patty.txt
```

```
Sandy and Nicki.txt

A:\>_
```

**WHAT'S HAPPENING?** You copied files to the **TRIP** directory. You then looked for those files that contained "SANDY" in the filename. The /R parameter searched all the directories on your disk. However, it showed you the entire path name. If you want to use commands such as REN, you need ***only*** the file name. The options preceded by the tilde allowed you to do so. The ***n*** forces the variable to expand to only the file name whereas the ***x*** forces the expansion only of the file extension. You can use these features to write a batch file that will allow you to precede any file name with any prefix you wish.

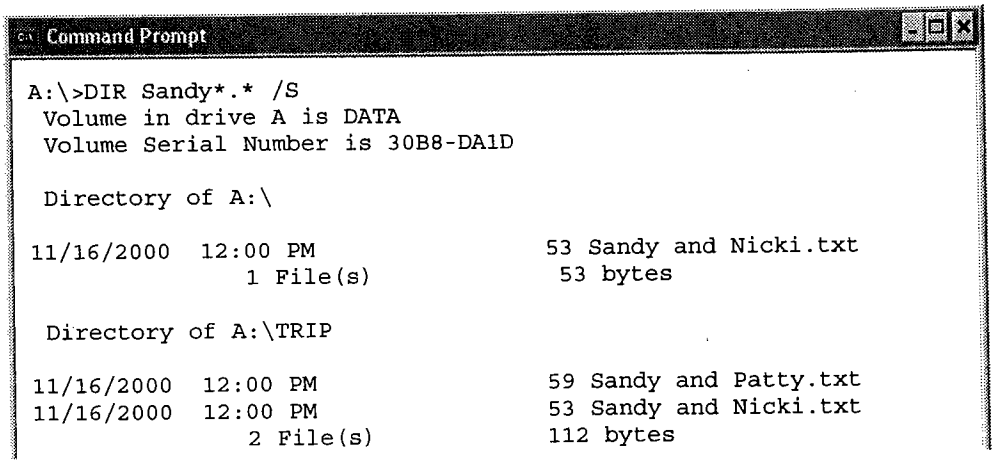
**13** In your batch file directory, create the following batch file called **PREFIX.BAT**. Remember that in a batch file, variable names need to be preceded by two percent signs. Use your name instead of "YourNameHere."

```
@ECHO OFF
REM YourNameHere
REM Purpose of batch file is to add a new prefix to any file name.
IF "%1"=="" GOTO MESSAGE
IF "%2"=="" GOTO MESSAGE2
FOR /R %%a IN (%2) DO REN "%%~a" "%1 %%~nxa"
GOTO END
:MESSAGE
ECHO You must include a prefix you wish to use.
ECHO Syntax is PREFIX prefix filename
GOTO END
:MESSAGE2
ECHO You must include a file name you wish to rename.
ECHO Syntax is PREFIX prefix filename
:END
```

**14** Be sure that the **BATCH** directory is in your path. You can use **ADD.BAT** to include it.

**15** Be sure you are in the root of A:\. Key in the following:

A:\>**DIR Sandy*.* /S** Enter



```
 Directory of A:\OLDER

11/16/2000  12:00 PM                59 Sandy and Patty.txt
11/16/2000  12:00 PM                53 Sandy and Nicki.txt
               2 File(s)            112 bytes

    Total Files Listed:
               5 File(s)            277 bytes
               0 Dir(s)         883,200 bytes free

A:\>_
```

**WHAT'S HAPPENING?** You have five files that have SANDY as the first characters in their names, one in the root of A and the others in a subdirectory called **TRIP** and the others in the OLDER directory. You are going to modify those files so that their names will be preceded by RBP.

**16** Key in the following: `A:\>`**PREFIX RBP SANDY*** Enter

**17** Key in the following: `A:\>`**DIR SANDY* /S** Enter

**18** Key in the following: `A:\>`**DIR RBP* /S** Enter

Command Prompt

```
A:\>PREFIX  RBP  SANDY*

A:\>DIR SANDY* /S
 Volume in drive A is DATA
 Volume Serial Number is 30B8-DA1D
File Not Found

A:\>DIR RBP* /S
 Volume in drive A is DATA
 Volume Serial Number is 30B8-DA1D

 Directory of A:\

11/16/2000  12:00 PM                53 RBP Sandy and Nicki.txt
               1 File(s)             53 bytes

 Directory of A:\TRIP

11/16/2000  12:00 PM                59 RBP Sandy and Patty.txt
11/16/2000  12:00 PM                53 RBP Sandy and Nicki.txt
               2 File(s)            112 bytes

 Directory of A:\OLDER

11/16/2000  12:00 PM                59 RBP Sandy and Patty.txt
11/16/2000  12:00 PM                53 RBP Sandy and Nicki.txt
               2 File(s)            112 bytes

     Total Files Listed:
               5 File(s)            277 bytes
               0 Dir(s)         883,200 bytes free

A:\>_
```

**WHAT'S HAPPENING?** You successfully renamed your files. You may also strip out specific fields in a text file. The FOR command also allows you to use the /F parameter. The /F parameter allows you to extract specific data from a text file. It lets you set the rules by which you will extract the data. The basic syntax is:

```
FOR /F "USERBACKQ=option TOKENS=list" %%variable IN ("set")
DO command
```

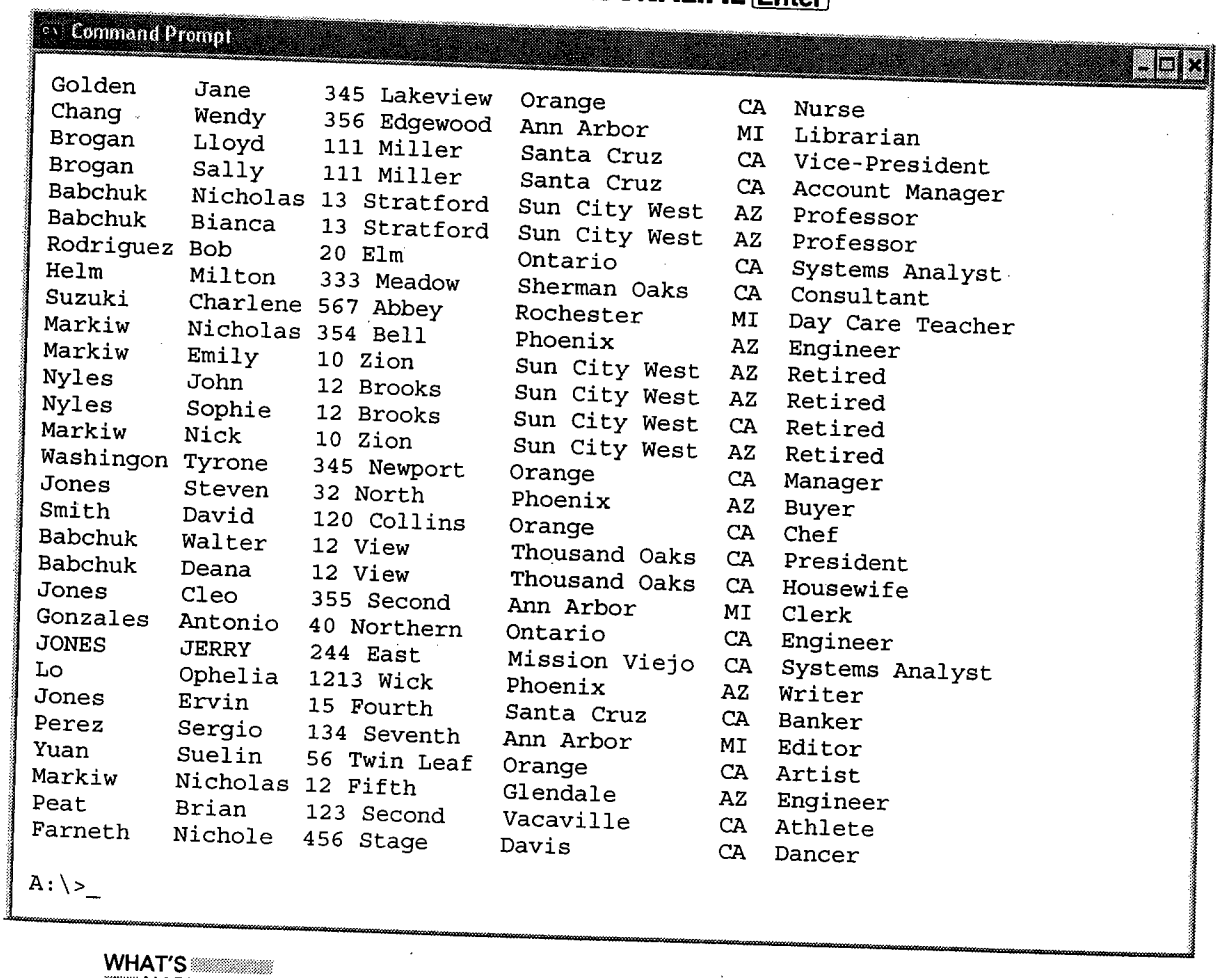
Using USERBACKQ, you may specify what delimiter you are going to use. If you do not specify a delimiter, then spaces or tabs are used. The TOKENS is a series of numbers telling the command which token on the text line is to be assigned the next %%variable.

**19** Create the following batch file called **PERSON.BAT** in the **BATCH** directory:

```
@ECHO OFF
FOR /F "tokens=1,2,7" %%a IN (%1) DO ECHO %%b %%a, %%c
```

**20** Key in the following, if necessary: `A:\BATCH>`**CD ** Enter

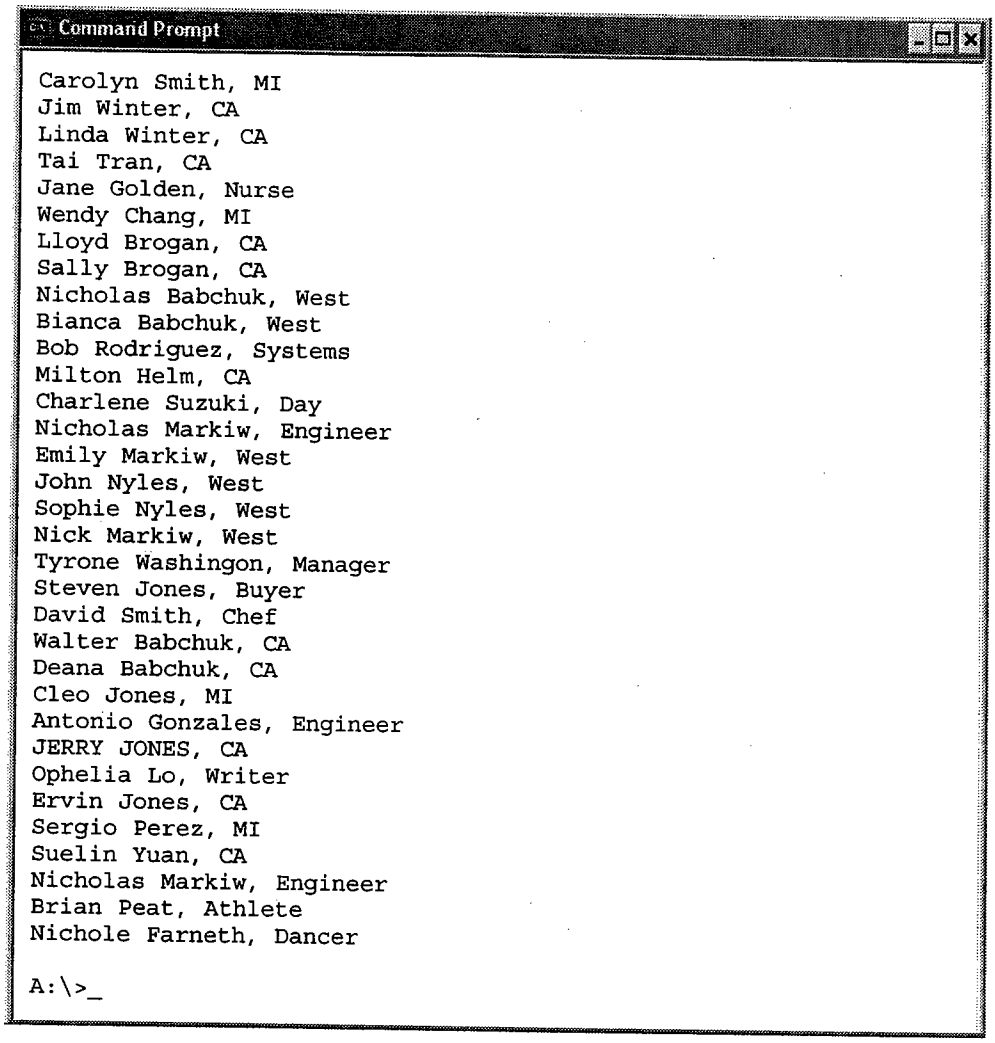
**21** Key in the following: `A:\>`**TYPE PERSONAL.FIL** Enter




**WHAT'S HAPPENING?** (Some of the file will scroll off your screen.) Here you have a text file. In the batch file you just wrote, you are going to use the /F option. The tokens that you specified are in the first, second, and seventh positions. In other words, you want the first name, last name, and profession extracted from this file. The last name

will be **a**, the first name will be **b**, and the profession will be **c**. These are assigned by the FOR command. You will display them so that you see first name, then last name, then profession.

**22** Key in the following: `A:\>`**PERSON PERSONAL.FIL** Enter

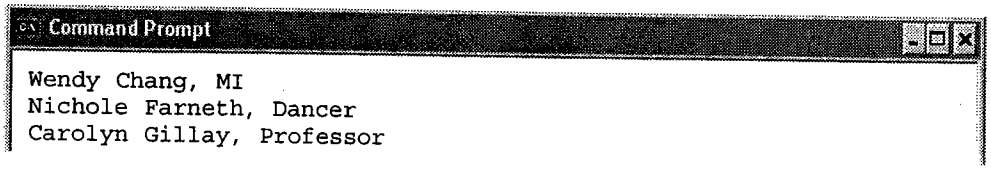



**WHAT'S HAPPENING?** As you can see, you extracted only two fields from the file. However, they are not all the fields you wanted. You can make your batch file more sophisticated by creating the output in sorted order.

**23** Edit the **PERSON.BAT** file in the **BATCH** directory as follows:

```
@ECHO OFF
SORT < %1 > %2
FOR /F "tokens=1,2,7" %%a IN (%2) DO ECHO %%b %%a, %%c
DEL %2
```

**24** Key in the following: `A:\>`**PERSON PERSONAL.FIL TEMP.FIL** Enter



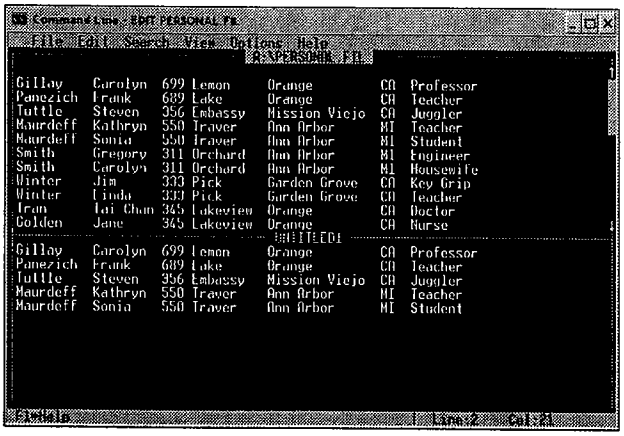
```
Jane Golden, Nurse
Antonio Gonzales, Engineer
Milton Helm, CA
Cleo Jones, MI
Ervin Jones, CA
JERRY JONES, CA
Steven Jones, Buyer
Ophelia Lo, Writer
Emily Markiw, West
Nicholas Markiw, Engineer
Nicholas Markiw, Engineer
Nick Markiw, West
Kathryn Maurdeff, MI
Sonia Maurdeff, MI
John Nyles, West
Sophie Nyles, West
Frank Panezich, Teacher
Brian Peat, Athlete
Sergio Perez, MI
Bob Rodriguez, Systems
Carolyn Smith, MI
David Smith, Chef
Gregory Smith, MI
Charlene Suzuki, Day
Tai Tran, CA
Steven Tuttle, CA
Tyrone Washingon, Manager
Jim Winter, CA
Linda Winter, CA
Suelin Yuan, CA

A:\>_
```

**WHAT'S HAPPENING?** You have a sorted list of people and their professions. However, there is still a problem. Many of the lines have the state listed instead of the profession. The batch file you wrote looks at a space as a delimiter. If you looked at the original data, the seventh position varies, depending on the number of words in the address.

| DATA | Yuan | Suelin | 56 | Twin Leaf | Orange | CA | Artist | Position |
|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

The problem here is the data. You can alter the data and choose a delimiter that will set off the fields as actual fields.

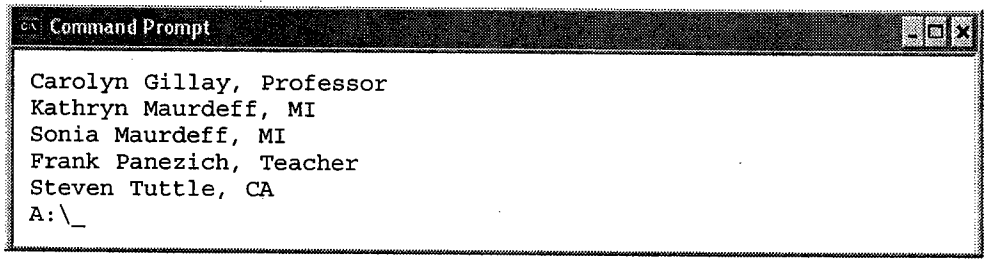
**25** Open the text editor. Split the screen. With the cursor in the top screen, open **PERSONAL.FIL**. Select the first five lines. Copy them into the bottom screen. Save the new file as **SHORT.FIL**.




**WHAT'S HAPPENING?** You have a smaller sample of data.

**26** Exit Edit and key in the following:

A:\>**PERSON SHORT.FIL TEMP.FIL** Enter

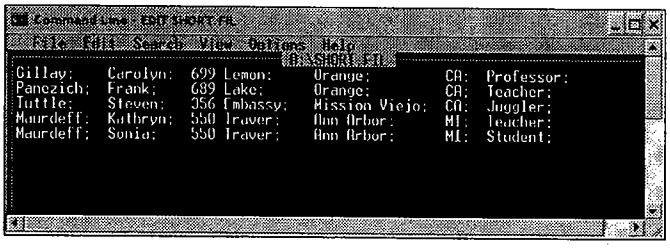



**WHAT'S HAPPENING?** You are still not getting the results that you want. You are going to edit the file, using semicolons to separate the fields.

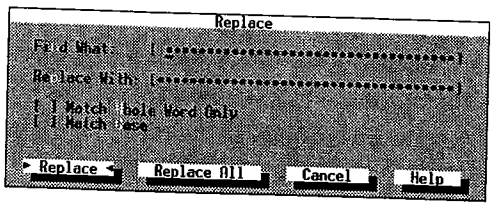
**27** Edit the **SHORT.FIL** file as follows:

**Gillay;Carolyn;699 Lemon;Orange;CA;Professor**
**Panezich;Frank;689 Lake;Orange;CA;Teacher**
**Tuttle;Steven;356 Embassy;Mission Viejo;CA;Juggler**
**Maurdeff;Kathryn;550 Traver;Ann Arbor;MI;Teacher**
**Maurdeff;Sonia;550 Traver;Ann Arbor;MI;Student**

*Note:* A simple way to do this is to place a semicolon at the end of each field, as shown below,




then use Replace on the Search menu to replace a space with nothing,




and Replace All and Save your file. Then put a space back between "Ann" and "Arbor" in the two occurrences of "AnnArbor" and between "Mission" and "Viejo" in the one occurrence of "MissionViejo."

**WHAT'S HAPPENING?** You now have a delimited file. Each field is set off with a semicolon. You could have used any character. If you had used commas, it would be considered a comma-delimited file. But you have to alter your batch file so that it knows that the character you selected is a semicolon. Note that tokens change from 1,2,7 to 1,2,6.

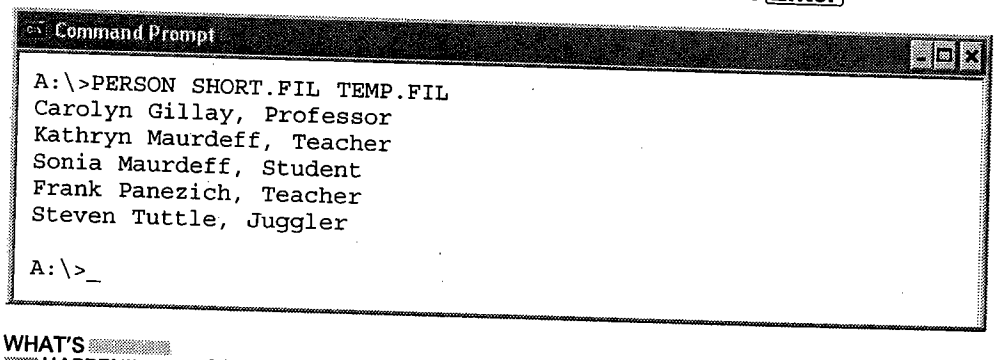
**28** Close **SHORT.FIL**, then edit **PERSON.BAT** so it looks as follows:

```
@ECHO OFF
SORT < %1 > %2
FOR /F "usebackq delims=; tokens=1,2,6" %%a IN (%2) DO ECHO
%%b %%a, %%c
DEL %2
```

*Note:* When entering the third line (beginning with "FOR"), do not press the Enter key until after %%c.

**WHAT'S HAPPENING?** The statement "usebackq delims" stated that you were going to use the ; as your delimiter. You also had to change the token. The last token is now 6, not 7.

**29** Key in the following: `A:\>`**PERSON SHORT.FIL TEMP.FIL** Enter




**WHAT'S HAPPENING?** Your data is delimited, and your output is displayed the way you want it.

# 11.30 The CALL Command

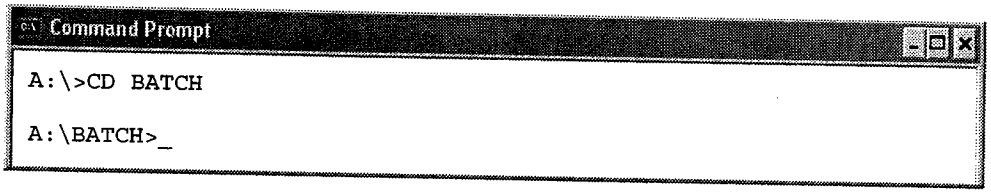
You sometimes need to be able to execute one batch file from within another. If the second batch file is the last line in the original batch file, there is no problem. The second batch file is "invoked" from the first batch file. But invoking a second batch file, executing it, and upon completion of the second file, returning to the first batch

file, is not so simple. When the operating system finishes executing a batch file, it returns control to the system level and never returns to the original batch file. There is a solution to this problem, the CALL command. It allows you to call (execute) another batch file and then return control to the next line in the first (calling) batch file.

## 11.31 Activity: Using CALL

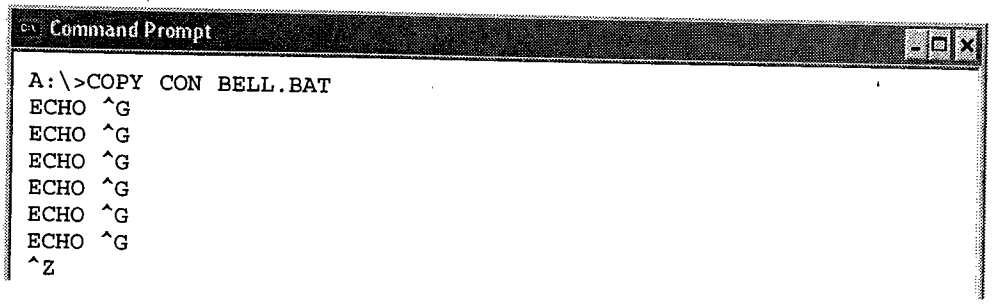
*Note:* The DATA disk should be in Drive A. The displayed prompt is A:\>. You have executed the command A:\BATCH\>**ADD A:\BATCH** at some point during the current Command Prompt session.

1 Key in the following: A:\>**CD BATCH** Enter




**WHAT'S HAPPENING?** Your default directory is now the **BATCH** subdirectory. One of the things that you can create is a "noise" to get a user's attention. You can use Ctrl + G to make a noise. All computers will not make a noise—most do. Although the noise you create sounds like a beep, it is referred to as a bell because in the ASCII character set Ctrl + G is labeled **BEL**. By using + G with ECHO you "ring a bell." Remember that when you see Ctrl + G, it means press the Ctrl key and the letter G. When you see F6, it means press the F6 function key. Today's computers are very fast—1.6 GHz is commonplace. The beep sound is very short, and may be difficult to hear. Therefore we will repeat the beep command six times, making it easier for you to hear the generated sound. Even if you cannot hear the sound, you will not get an error message.

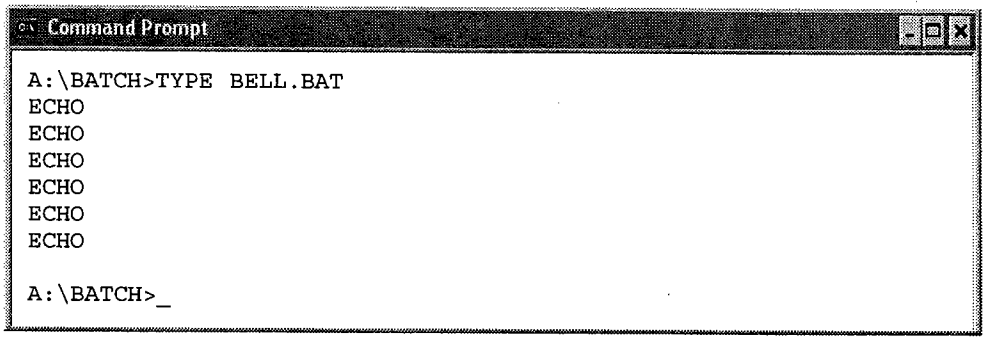
2 Key in the following: A:\BATCH>**COPY CON BELL.BAT** Enter
**ECHO** Ctrl + G Enter
**ECHO** Ctrl + G Enter
**ECHO** Ctrl + G Enter
**ECHO** Ctrl + G Enter
**ECHO** Ctrl + G Enter
**ECHO** Ctrl + G Enter
F6 Enter



```
        1 file(s) copied.

A:\BATCH>_
```

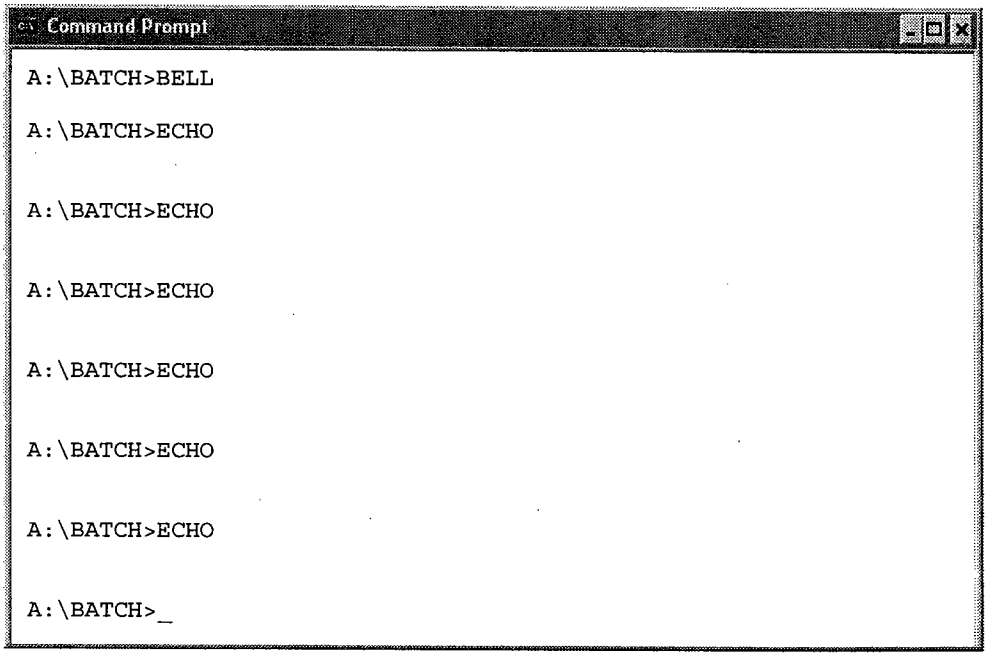
WHAT'S HAPPENING? Now that you have written **BELL.BAT**, you can execute it.

**3** Key in the following: `A:\BATCH>`**TYPE BELL.BAT** Enter




WHAT'S HAPPENING? Even when you type the file, you hear the bell.

**4** Key in the following: `A:\BATCH>`**BELL** Enter



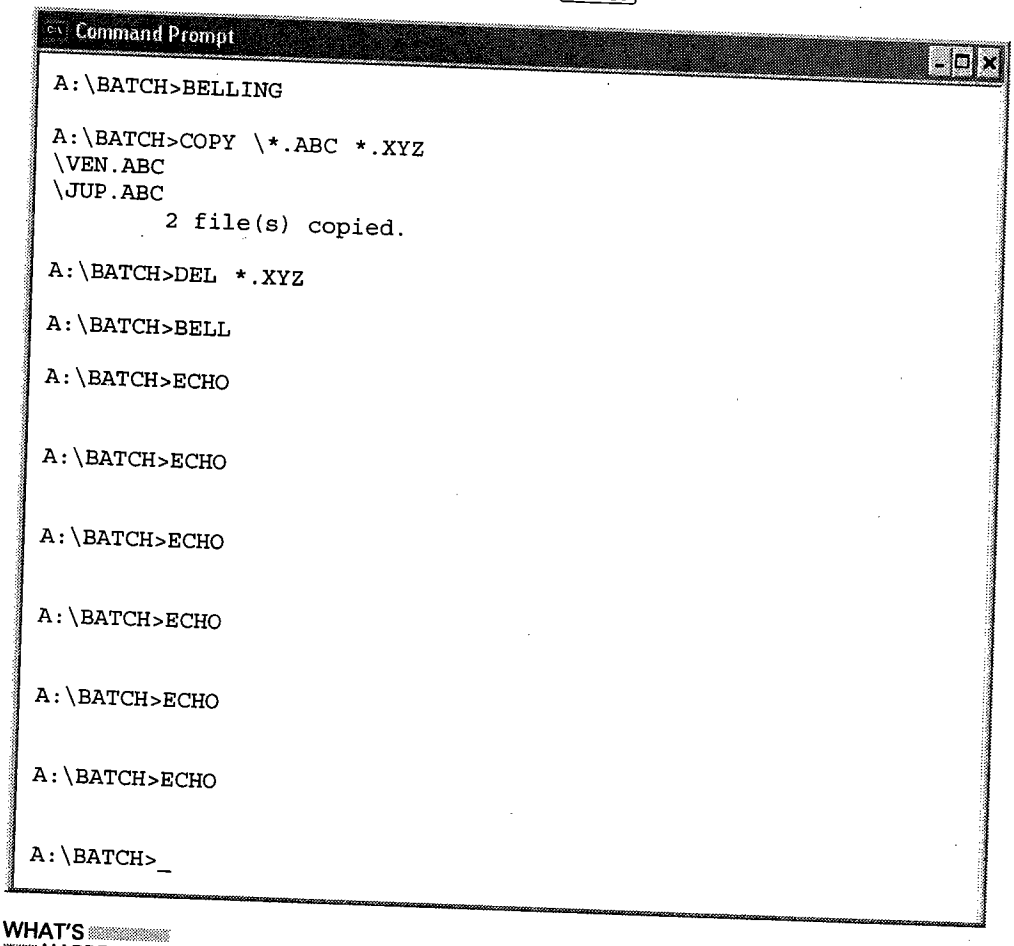

WHAT'S HAPPENING? You should have heard the bell beeping or clicking as you ran the program. Notice, you heard 12 "beeps" instead of only 6. When the command was read, the "bell" sounded, when the command was executed, the "bell" sounded again.

**5** Use any text editor to create and save a batch file called **BELLING.BAT** in the **BATCH** directory. Key in the following:
**COPY *.ABC *.XYZ** Enter
**DEL *.XYZ** Enter

**BELL** Enter
(Be sure to hit Enter after **BELL**.)

WHAT'S HAPPENING? This batch file will copy the .ABC files to the BATCH subdirectory, delete them, and then ring a bell.

6 Key in the following: `A:\BATCH>`**BELLING** Enter

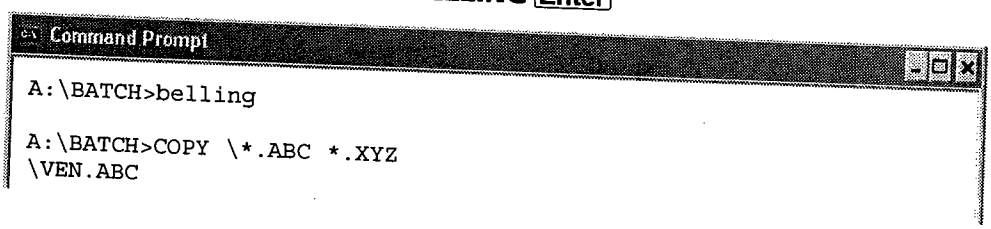



WHAT'S HAPPENING? You called one batch file from another. You ran **BELLING**. Its last line was BELL. **BELL** was called and it executed as expected. However, perhaps you would like the bell to sound before the files are deleted.

7 Edit and save **BELLING.BAT** to look as follows:

```
COPY \*.ABC *.XYZ
BELL
REM You are about to delete the *.XYZ files. Are you sure?
PAUSE
DEL *.XYZ
```

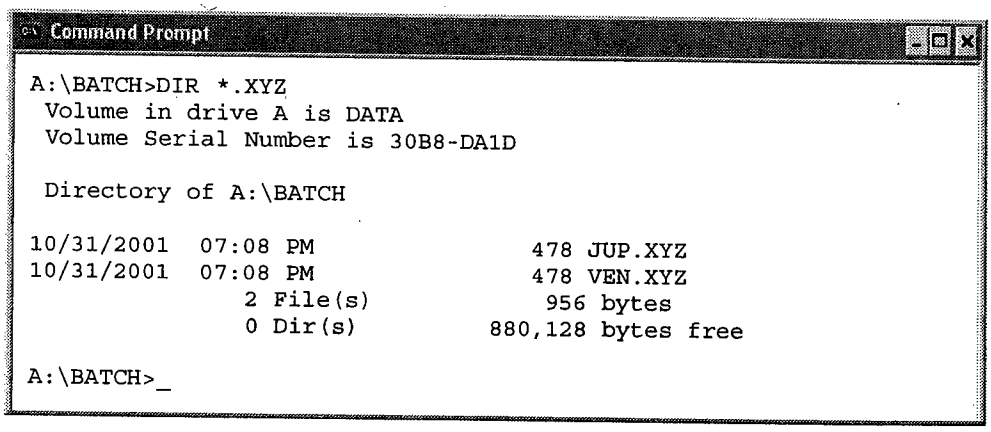
8 Key in the following: `A:\BATCH>`**BELLING** Enter



```
\JUP.ABC
        2 file(s) copied.

A:\BATCH>BELL

A:\BATCH>ECHO


A:\BATCH>ECHO


A:\BATCH>ECHO


A:\BATCH>ECHO


A:\BATCH>ECHO


A:\BATCH>ECHO


A:\BATCH>_
```

WHAT'S HAPPENING? When the **BELLING** batch file reached the line BELL, it called and executed **BELL**. It turned control over to **BELL.BAT**. Once **BELL.BAT** had control, it never returned to **BELLING**. Since it never returned to **BELLING.BAT**, your ***.XYZ** files were not deleted, nor did you see a message.

**9** Key in the following: `A:\BATCH>`**DIR *.XYZ** Enter

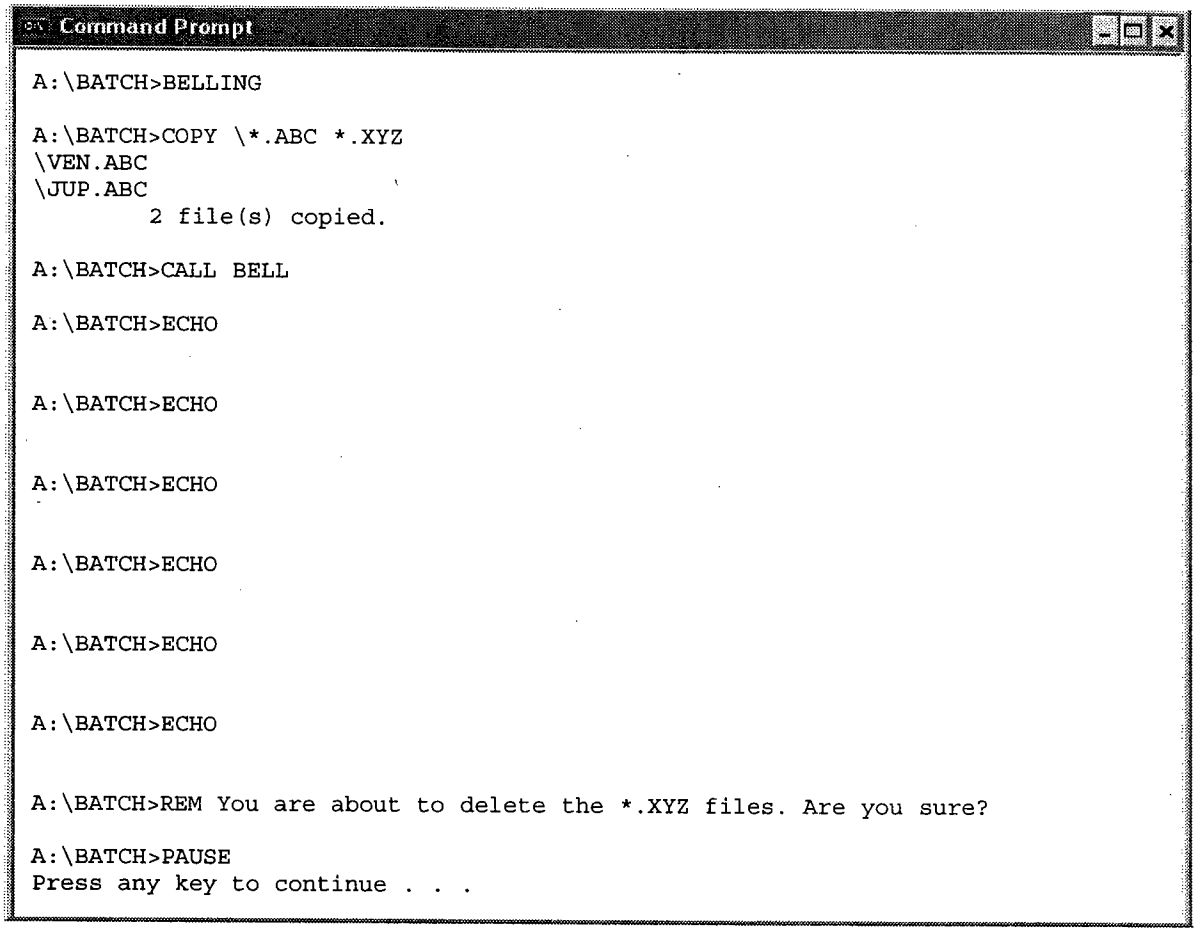



WHAT'S HAPPENING? Indeed, the files are there. This is why you need the CALL command. CALL will process the batch file **BELL**, but it will then return control to **BELLING** so that the other commands in **BELLING** can be executed.

**10** Edit and save **BELLING.BAT** to look as follows:

```
COPY \*.ABC *.XYZ
CALL BELL
REM You are about to delete the *.XYZ files. Are you sure?
PAUSE
DEL *.XYZ
```

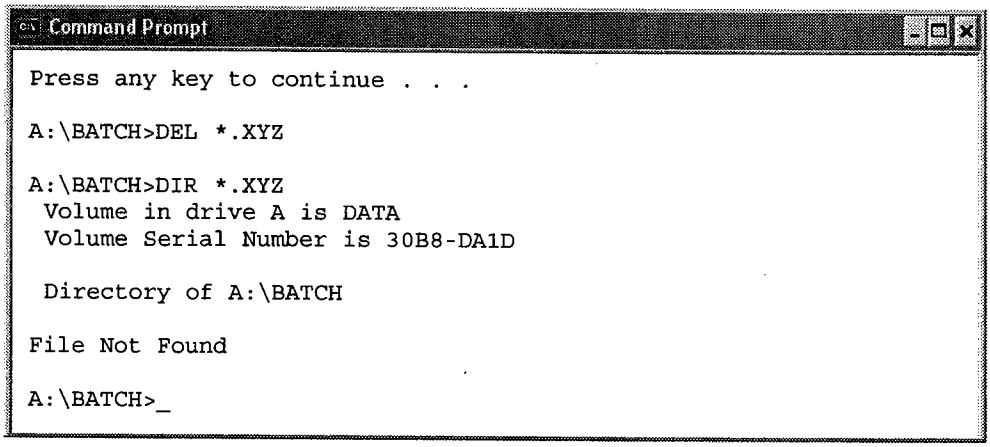
**11** Key in the following: `A:\BATCH>`**BELLING** Enter




**WHAT'S HAPPENING?** Since you added CALL in front of BELL, the bell sounded. Once **BELL** was finished executing, it passed control back to **BELLING** so that the next commands could be executed.

**12** Press Enter

**13** Key in the following: `A:\BATCH>`**DIR *.XYZ** Enter




**WHAT'S HAPPENING?** The CALL command worked as promised. Your **.XYZ** files are gone. A more practical example of using CALL can be seen in the next group of batch files you are going to write. You may find, as you move among directories,

that you want to go "home" again. In other words, you want to return to the directory where you were previously. You can create a series of batch files that will remember the directory you were in and return you to it.

*Note:* In the next file, it is important to press the F6 key immediately after the command *before* you press Enter. Do not press Enter, then F6. Create the file in the BATCH directory.

**14** Use COPY CON to create and save **HOME.DAT** in the BATCH directory. Key in the following:

**COPY CON HOME.DAT** Enter
**SET HOME=**F6 Enter

**15** Use any editor to create and save a batch file called **HOMETO.BAT** in the BATCH directory. Key in the following:

**COPY A:\BATCH\HOME.DAT A:\BATCH\HOMESAVE.BAT** Enter
**CD >> A:\BATCH\HOMESAVE.BAT** Enter
**CALL HOMESAVE.BAT** Enter
**DEL A:\BATCH\HOMESAVE.BAT**

WHAT'S HAPPENING? The **HOME.DAT** data file you created will create an environmental variable called HOME. The batch file **HOMETO.BAT** that you just created will be used to set the environmental variable to wherever you want HOME to be. The batch file, line by line, breaks down as follows:

*Line 1:* Copies the contents of the data file to the batch file. **HOME.DAT** now contains SET HOME=. **HOMESAVE.BAT** now contains SET HOME=.

*Line 2:* Takes whatever directory you are in and appends it to **HOMESAVE.BAT**. If your current directory is **BATCH**, **HOMESAVE.BAT** now has the contents of SET HOME=A:\BATCH.

*Line 3:* Executes the batch file. **HOMESAVE.BAT** now executes and sets the variable of **HOME** to **A:\BATCH**.

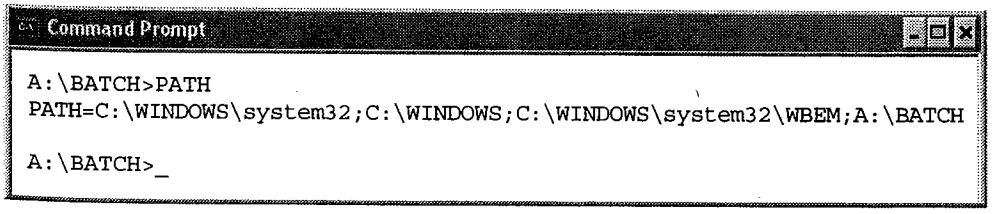
*Line 4:* Deletes the batch file. Now that you have set the environmental variable, you no longer need the batch file **HOMESAVE.BAT**.

**16** Use any editor to create and save a batch file called **HOME.BAT** in the BATCH directory that contains the following:

**CD %HOME%** Enter

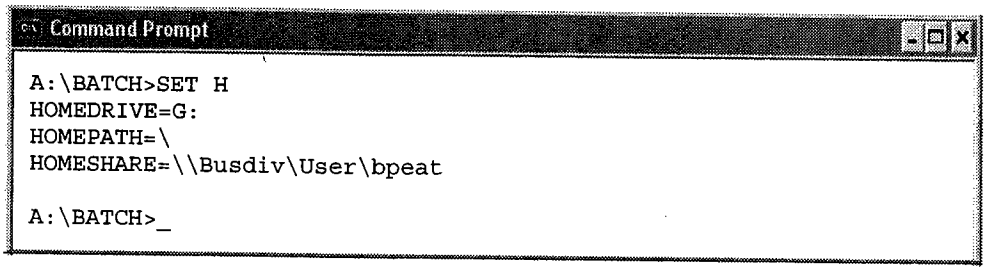
WHAT'S HAPPENING? The batch file **HOME.BAT** will change your directory to whatever value is in the environmental variable **HOME** at the time the batch file is executed. In order for this procedure to work correctly, you must include A:\BATCH in your path statement.

**17** Key in the following: **PATH** Enter



**WHAT'S HAPPENING?** In this example, A:\BATCH is indeed included in the path. If it is not included in your path, run ADD A:\BATCH before proceeding.

**18** Key in the following: `A:\BATCH>`**SET H** Enter

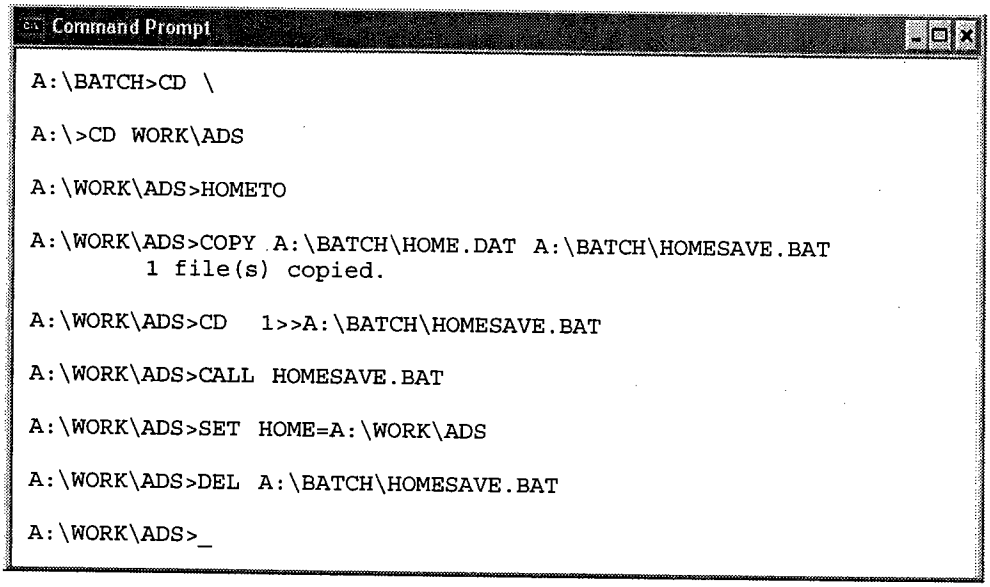



**WHAT'S HAPPENING?** A:\BATCH is included in the path. There is no environmental variable HOME displayed. You are now ready to test "going home."

**19** Key in the following: `A:\BATCH>`**CD ** Enter

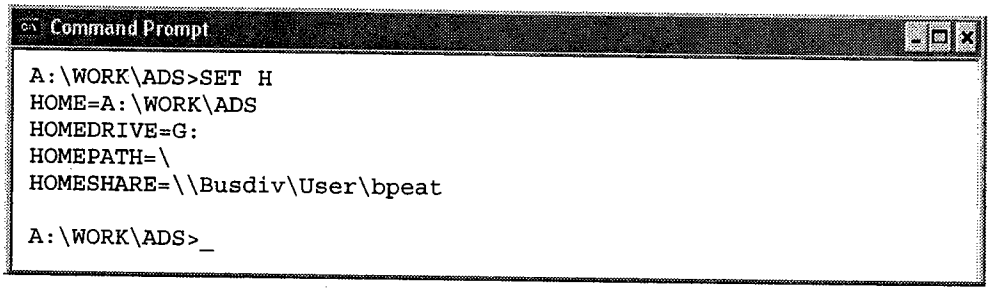
**20** Key in the following: `A:\>`**CD WORK\ADS** Enter

**21** Key in the following: `A:\WORK\ADS>`**HOMETO** Enter



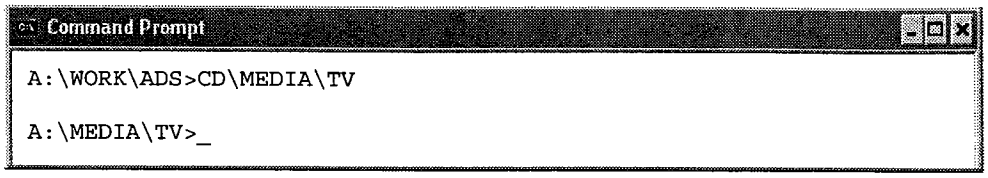

**WHAT'S HAPPENING?** You wanted the subdirectory **\WORK\ADS** to be your home directory so you keyed in **HOMETO**.

**22** Key in the following: `A:\WORK\ADS>`**SET H** Enter



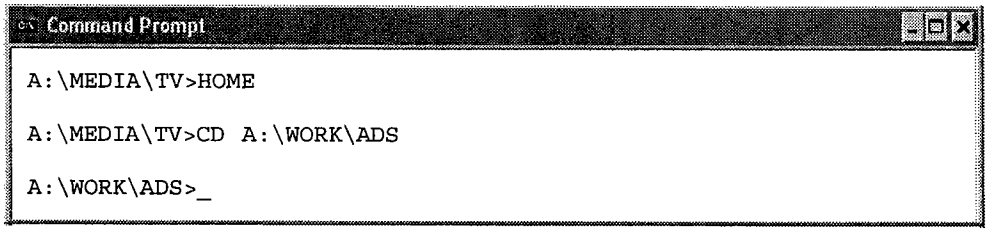
WHAT'S HAPPENING? You now have an environmental variable named HOME whose value is **A:\WORK\ADS**. You will go to another directory and then return to **A:\WORK\ADS** by using the HOME command you just wrote.

**23** Key in the following: `A:\WORK\ADS>`**CD \MEDIA\TV** Enter




WHAT'S HAPPENING? Your default directory is now **A:\MEDIA\TV**. You want to return home, which is **A:\WORK\ADS**.

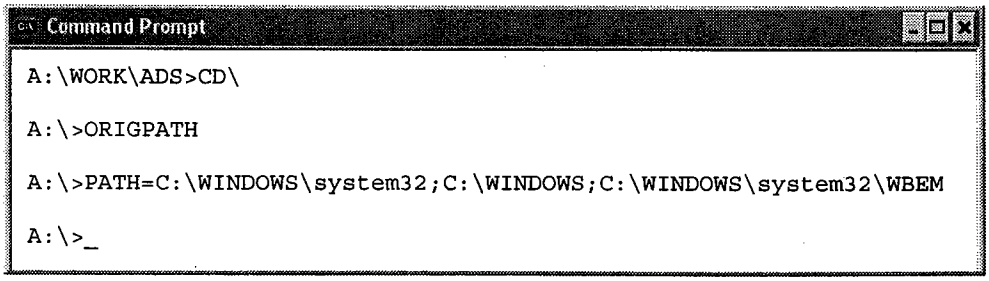
**24** Key in the following: `A:\MEDIA\TV>`**HOME** Enter




WHAT'S HAPPENING? You went home—home being the value set in the HOME variable. At any time during the Command Prompt session, you can change the value of home by running HOMETO while in the directory you want home to be, and you can return to it at any time by simply keying in **HOME**.

**25** Key in the following: `A:\WORK\ADS>`**CD ** Enter

**26** Key in the following: `A:\>`**ORIGPATH** Enter




WHAT'S HAPPENING? You have returned to the root directory of the DATA disk and reset the default path.

# Chapter Summary

1. You may substitute a double colon (::) for the REM statement.
2. To place a blank line in a batch file, use the ECHO command followed immediately by a period (ECHO.).
3. The GOTO command used in conjunction with a label creates a loop. The GOTO will process the command following the label.

4. The label in a batch file is not a command, but identifies a location in the batch file.
5. The SHIFT command shifts over command line parameters to the left one position at a time.
6. The SHIFT command is typically used in conjunction with GOTO and a label.
7. The IF command will test for some logical condition. If the condition is true, the command will be processed. If the condition is false, the batch file will fall through to the next line of the batch file.
8. The IF command can test whether or not two character strings are identical.
9. The IF command can test whether or not a file exists.
10. The IF command checks ERRORLEVEL.
11. You may also use IF with NOT. The IF NOT command will test for a NOT condition. If a condition is not true, then the command will process. If the command is true, then the batch file will fall through to the next line in the batch file.
12. You can test for a null value by using quotation marks, a word, or backslashes.
13. To use IF EXIST and to test for the existence of a subdirectory, you may use IF %1\NUL.
14. Many programs set an exit code when finished executing. The IF ERRORLEVEL in a batch file will test if an exit code is equal to or greater than the one in the test.
15. When you use IF ERRORLEVEL in a batch file, the codes must be listed in descending order.
16. When you use IF NOT ERRORLEVEL in a batch file, the codes must be listed in ascending order.
17. You may write programs with DEBUG to set exit codes.
18. You can write programs directly in DEBUG, or you may write a script file that supplies input for DEBUG to create a program.
19. The environment is an area in memory where the operating system leaves messages to itself, including the path and the prompt values.
20. You can create environmental variables that can be used in batch files. When using the variable in a batch file, the syntax is *%variable%*.
21. The SET command lets you view what is currently in the environment.
22. Environmental variables set in other batch files or at the command line remain in effect only during the current Command Prompt session. When the Command Prompt window is closed, the variable disappears.
23. The DIRCMD command can be used to preset the DIR command parameters and switches.
24. The FOR...IN...DO command allows repetitive processing. The command will execute some command for every value in a specified set.
25. When you use FOR...IN...DO at the command line, the syntax is FOR *%variable* IN (*set*) DO *command* [*command-parameter*].
26. The /R when used with FOR...IN...DO allows recursive processing. Recursive means that the command will search all the directories.
27. Using the /F parameter with FOR...IN...DO allows you to select specified text using delimiters that you can set.
28. The CALL command allows you to call one batch file from another. When the second batch file is finished executing, it returns control to the first batch file.

# Key Terms

debug
EOF (end-of-file) mark
exit code
conditional processing
environment
environmental variable
expression
loop
null value
recursive
scan code
script file
variable

# Discussion Questions

1. What is the function of the REM, ECHO, and PAUSE commands?
2. What happens in a batch file if ECHO is set to OFF?
3. What happens in a batch file if you precede the ECHO OFF switch with @?
4. What is a NUL device? Why would you use a NUL device?
5. How can you place a blank line in a batch file?
6. How can you create a loop in a batch file? How can you stop a loop from processing in a batch file?
7. What is the purpose and function of the GOTO command?
8. What is a label in a batch file?
9. What is the purpose and function of the SHIFT command?
10. Why is it useful to shift parameters?
11. How can you determine whether or not a file exists?
12. What is the purpose and function of the IF command?
13. Give the syntax of the IF command and explain each part of the syntax.
14. What does it mean to test for a null value?
15. How can you test for a null value? Why would you test for a null value?
16. What is the purpose and function of the IF EXIST/IF NOT EXIST command?
17. Explain the purpose and function of the IF ERRORLEVEL command.
18. What is a script file? How can you create one?
19. What is a scan code?
20. Give the syntax of the SET command and explain each part of the syntax.
21. Explain the purpose and function of the DIRCMD environmental variable.
22. What is the purpose and function of the FOR...IN...DO command?
23. Name two parameters you can use with the FOR...IN...DO command.
24. Describe the purpose of those parameters.
25. Give the basic syntax of the FOR...IN...DO command and explain each part of the syntax.
26. Explain the purpose and function of the CALL command.

# True/False Questions

For each question, circle the letter T if the statement is true and the letter F if the statement is false.

T F 1. The SHIFT command may be used only in batch files.
T F 2. Environmental variables set at the command line remain in effect until the computer is turned off.
T F 3. You may alter the way DIR displays by changing the values in DIRCMD.

T F 4. Keying in SET at the command line shows the correct date and time.

T F 5. In order to run a batch file from another batch file and then return to the original batch file, you must use the CALL command.

# Completion Questions

Write the correct answer in each blank space.

6. In a batch file, to see whether or not the file called MY.FIL is a valid file, you must have the line IF _____ MY.FIL.
7. In a batch file, the line IF "%1"= ="" GOTO END is testing for a(n) _____.
8. You can use the _____ parameter or switch with the FOR command to look at files in different directories.
9. When you key in SET THIS=DIR *.TXT, you are setting a(n) _____.
10. When you use the GOTO command, you must also use a(n) _____.

# Multiple Choice Questions

For each question, write the letter for the correct answer in the blank space.

____ 11. If ECHO. (ECHO followed immediately by a period) is entered into a batch file, it will
   a. display the status of ECHO.
   b. create a blank line.
   c. do nothing.
   d. cause the following command not to execute.

____ 12. ECHO %PATH% (keyed in at the command line) will
   a. produce an error message.
   b. display the contents of the file named PATH.
   c. display the path for the current session.
   d. display the environmental variables set in the current session.

____ 13. The following command in a batch file—IF %1none==none GOTO TOP—will
   a. test for a null value.
   b. test for the existence of a file.
   c. test for ERRORLEVEL.
   d. all of the above

____ 14. A valid batch file label consists of a
   a. percent sign (%) and text chosen by the user.
   b. colon (:) and text chosen by the user.
   c. double colon (::) and text chosen by the user.
   d. any of the above

____ 15. So that %3 becomes %2, and %2 becomes %1, you must use the _____ command in a batch file.
   a. ROTATE %1
   b. SHIFT %1
   c. SHIFT
   d. none of the above

# Homework Assignments

*Note 1:* Place the HOMEWORK disk in Drive A. Be sure to work on the HOMEWORK disk, not the DATA disk.

*Note 2:* The homework problems will assume Drive C is the hard disk and the HOMEWORK disk is in Drive A. If you are using another drive, such as floppy drive B or hard drive D, be sure and substitute that drive letter when reading the questions and answers.

*Note 3:* Test all of your batch files before submitting them to be sure they work correctly.

*Note 4:* To save a file under a new name with Edit, press Alt + F and choose **Save As**.

*Note 5:* There can be more than one way to write a batch file. If your batch file works correctly, it is most likely written correctly.

## Setup

1. Place the HOMEWORK disk in Drive A.
2. Create a **BATCH** subdirectory on the HOMEWORK disk and move any batch files from the root directory into the **BATCH** subdirectory.
3. Move **GO.BAT** and **NAME.BAT** from the **BATCH** subdirectory back to the root of the HOMEWORK disk.
4. Create a subdirectory called **CHAP11** off the root directory of the HOMEWORK disk.

## Problem Set I

### Problem A

**A-a** Make **CHAP11** the default directory.

**A-b** Create and save a batch file in the **CHAP11** subdirectory called **DELBAK.BAT** that has the following line in it:
**FOR %%h IN (*.BAK) DO IF EXIST %%h DEL %%h**

____ 1. The DELBAK.BAT file will
   a. delete any file in the default drive and directory.
   b. delete any file in the default drive and directory with the .BAK extension.
   c. first check whether or not any files with the .BAK file extension exist in the current drive and directory.
   d. both b and c

____ 2. If you wanted to use the previous batch file command on the command line, what would you have to do?
   a. change %%h to %h
   b. change %%h to %h%
   c. change *.BAK to %h%
   d. none of the above

**A-c** Create the following batch file in the **CHAP11** subdirectory and save it as **LIST.BAT**:

```
@ECHO OFF
ECHO.
FOR %%v IN (%PATH%) DO ECHO %%v
ECHO.
ECHO.
```

____ 3. The LIST.BAT file will
   a. display the directory names in the current path.
   b. use an environmental variable.
   c. both a and b
   d. neither a nor b

## Problem B

*Note:* If you are in a lab environment or if your user name has spaces in it, there may not be a correct answer listed for questions 4 and 5.

**B-a** Be sure that **CHAP11** is the default directory on the HOMEWORK disk.

**B-b** In the **CHAP11** subdirectory, create and save a batch file called **PASSING.BAT** that has the following lines in it:

```
@ECHO OFF
FOR %%a IN (%USERNAME%) DO IF "%1"=="%%a" GOTO DISPLAY
:NOT
ECHO You used %1. This is not your correct login name.
GOTO END
:DISPLAY
DIR %homepath% /p
:END
```

**B-c** Execute **PASSING.BAT** using your own logon name.

____ 4. When you used PASSING.BAT you
   a. displayed the files in the current default directory.
   b. displayed the files in the root of the HOMEWORK disk.
   c. displayed the files in the root of Drive C.
   d. displayed the message, "You used Bette. This is not your correct login name."

**B-d** Execute **PASSING.BAT** using Bette.

____ 5. When you used PASSING.BAT with Bette, you
   a. displayed the files in the current default directory.
   b. displayed the files in the root of the HOMEWORK disk.
   c. displayed the files in the root of Drive C.
   d. displayed the message, "You used Bette. This is not your correct login name."

## Problem C

**C-a** Be sure that **CHAP11** is the default directory on the HOMEWORK disk.

*Note:* The following batch file is called **SWAP.BAT**. Its purpose is to allow the user to swap file names between two existing files. If the user keyed in **SWAP MY.OLD MY.NEW**, the file **MY.OLD** would then be named **MY.NEW** and the file **MY.NEW** would then be named **MY.OLD**. However, this batch file has a problem and does not work properly.

**C-b** In the **CHAP11** subdirectory, create and save a batch file called **SWAP.BAT** that has the following lines in it:

```
@ECHO OFF
IF NOT EXIST %1 GOTO NOFILE1
IF NOT EXIST %2 GOTO NOFILE2
REN %1 HOLD
REN %2 %1
REN HOLD %2
GOTO END
:NOFILE1
ECHO Cannot find file %1
:NOFILE2
ECHO Cannot find file %2
:END
```

**C-c** If you do not have a subdirectory called **\SPORTS**, create it now. Then copy any file with a **.RED** extension from the **WUGXP** subdirectory to the **\SPORTS** subdirectory on the HOMEWORK disk.

**C-d** Key in the following:

`A:\CHAP11>`**SWAP \SPORTS\LEFT.RED \SPORTS\RIGHT.RED** Enter

____ 6. When you used **SWAP.BAT**, you received an error message and **SWAP.BAT** did not work. Why?

a. You cannot name a file **HOLD**.
b. The file **\SPORTS\LEFT.RED** does not exist.
c. The syntax of the REN command is incorrect.
d. You were immediately sent to the :END label.

____ 7. Did any file get renamed in the **\SPORTS** directory?

a. yes
b. no

# Problem Set II

*Note 1:* The HOMEWORK disk is in Drive A and A:\> is displayed as the default drive and the default directory. All work will occur on the HOMEWORK disk.

*Note 2:* If **NAME.BAT**, **MARK.FIL**, **GETYN.COM**, and **GO.BAT** are not in the root directory, copy them from the **WUGXP** directory before proceeding.

**1** Key in the following: `A:\>`**NAME** Enter

**2** Here is an example to key in, but your instructor will have other information that applies to your class. Key in the following:

**Bette A. Peat** Enter (*Your* name goes here.)
**CIS 55** Enter (*Your* class goes here.)
**T-Th 8–9:30** Enter (*Your* day and time go here.)
**Chapter 11 Homework** Enter
**Problem A** Enter

3 Press F6 Enter. If the information is correct, press **Y** and you are back to A:\>.

4 Begin a new Command Prompt session by closing any existing Command Prompt windows and opening a new window.

5 Make A:\> the default directory.

6 Key in the following: `A:\>`**PATH > MYPATH.BAT** Enter

## Problem A

**A-a** Make **CHAP11** the default directory.

**A-b** The following batch file is called **SETPATH.BAT**. Its purpose is to allow the user to add a path to either the front or the back of the existing path. Any number of subdirectory names can be added. When the user keys in **SETPATH /F subdirectory1 subdirectory2** or **SETPATH /f subdirectory1 subdirectory2**, the /F or /f tells the batch file to add the subdirectory name in front of the existing path. If the SETPATH command is keyed in without /F or /f, each subdirectory listed will be added to the end of the path.

**A-c** Analyze the following batch file:

```
IF "%1"=="" GOTO DEFAULT
IF "%1"=="/f" GOTO FRONT
IF "%1"=="/F" GOTO FRONT
:ADD
IF "%1"=="" GOTO END
PATH= %PATH%;%1
SHIFT
GOTO ADD
:FRONT
SHIFT
IF "%1"=="" GOTO END
PATH=%1;%PATH%
GOTO FRONT
:DEFAULT
PATH=C:\WINDOWS\SYSTEM32
:END
```

**A-d** In the **CHAP11** subdirectory, create and edit a batch file called **SETPATH.BAT** based on the model above.

**A-e** Document it with your name and date, and explain the purpose of the batch file.

**A-f** Copy the file called **REPLY.COM** that you wrote in Activity 11.XX from the BATCH directory of the DATA disk to the **CHAP11** directory. Then begin the

batch file so that you may give the user the opportunity to choose help on how to use this command. The user will press **Y** or **y** if they want help or **N** or **n** if they do not want help. If the user chooses help, provide help on syntax as well as a description on how to use **SETPATH.BAT**. After displaying the description, the user should exit from the batch file. *Note:* The scan code value for Y is 89 and for y is 121. The scan code value for N is 78 and for n is 110.

**A-g** Change the :DEFAULT section so that it calls whatever your original path was, rather than **PATH C:\WINDOWS\SYSTEM32** . (*Hint*: Remember, **MYPATH.BAT** is in the root directory of the HOMEWORK disk, created in Problem A.)

**A-h** Key in the following: `A:\CHAP11>`**CD** Enter

**A-i** Key in the following: `A:\>`**GO NAME.FIL CHAP11\SETPATH.BAT** Enter Follow the instructions on the screen.

**A-j** In Notepad, click **File**. Click **Print**. Click the **Print** button. Click **File**. Click **Exit**.

## Problem B

**B-a** Make **CHAP11** the default directory.

**B-b** Use **SETPATH**, created in Problem A, to add the **A:\CHAP11** and the **A:\BATCH** subdirectories to the path.

**B-c** Using as a model **MY.BAT** from Activity 11.XX, create and save a batch file called **DIRS.BAT** in the **CHAP11** subdirectory.

**B-d** Document it with your name and date and explain the purpose of the batch file.

**B-e** Have the following choices in **DIRS.BAT**:

- Look at directories only, arranged alphabetically by name in the root directory. (Call this Root Directory Only.)
- Look at directories only, arranged alphabetically by name on the entire disk beginning with the root directory. (Call this All Directories.)
- Look at file names only—no dates, no directory information—on the root directory in reverse alphabetic order. (*Hint:* Remember /b.) (Call this **File Names Only**.)
- Look at files only in the root directory arranged by date/time. (Call this **Files by Date/Time**.) *Note:* Do not use date alone in an ECHO line. If you do, you are asking that the DATE command be executed.
- The scan code value for 1 is 49, 2 is 50, 3 is 51, and 4 is 52. If you want to use other key choices, see Appendix C for the scan code values.
- Remove any other choices.
- Be sure no command lines are displayed, only the results of the commands.

**B-f** Key in the following: `A:\CHAP11>`**CD** Enter

**B-g** Use Edit to modify **NAME.FIL**, changing **Problem A** to **Problem B**.

**B-h** Key in the following: `A:\>`**GO NAME.FIL CHAP11\DIRS.BAT** Enter
Follow the instructions on the screen.

**B-i** In Notepad, click **File**. Click **Print**. Click the **Print** button. Click **File**. Click **Exit**.

## Problem C

**C-a** Make **CHAP11** the default directory.

**C-b** Check the path to see if **CHAP11** is in the path. If not, use **SETPATH**, created in Problem A, to add the A:**CHAP11** subdirectory to the path. Add the **A:\BATCH** subdirectory to the path as well.

*Note:* The following batch file is called **WORD.BAT**. Its purpose is to allow the user to key in WORD *filename.ext*. If the file already exists, the user is immediately taken into Edit with the named file. If, however, the file is a new file, the user will be told to key in WORD and the new file name at the command line. This batch file, as written, has a fatal flaw. Every time the user keys in WORD *newfile.nam* he or she is kicked out of the batch file.

**C-c** Analyze the following batch file:

```
:START
IF EXIST %1 GOTO PROCEED
IF NOT EXIST %1 GOTO NEWFILE
:PROCEED
EDIT %1
GOTO END
:NEWFILE
ECHO This is a new file, %1. If you
ECHO wish to create a new file, you must
ECHO key in WORD and the new file name at the command line.
ECHO The syntax is:
ECHO WORD new.fil
:END
```

**C-d** Edit and save a batch file called **WORD.BAT** in the **CHAP11** subdirectory based on the above model. Document it with your name and date and explain the purpose of the batch file.

- If the user keys in WORD with no file name, the batch file should tell the user what was done wrong and return the user to the system level.
- If the user keys in WORD filename.ext and it is an existing file, the file should go directly into Edit using that file name. However, if the file does not exist, offer the user two choices—either to not create a new file and exit the batch file or to create a new file and be taken back to the PROCEED label. You will use REPLY and ERRORLEVEL so this choice can be made. *Note:* The scan code value for Y is 89 and for y is 121. The scan code value for N is 78 and for n is 110.

**C-e** Key in the following: `A:\CHAP11>`**CD ** Enter

**C-f** Use Edit to modify **NAME.FIL**, changing **Problem B** to **Problem C**.

**C-g** Key in the following: `A:\>`**GO NAME.FIL CHAP11\WORD.BAT** Enter
Follow the instructions on the screen.

**C-h** In Notepad, click **File**. Click **Print**. Click the **Print** button. Click **File**. Click **Exit**.

## Problem D: Challenge Assignment

The following batch file change is difficult, so do it only if you want a challenge.

**D-a** Make **CHAP11** the default directory.

**D-b** Check the path to see if **CHAP11** is in the path. If not, use **SETPATH**, created in Problem A, to add the **CHAP11** subdirectory to the path.

*Note:* The following batch file is called COMPILE.BAT. Its purpose is to allow the user to make a list of all the files in all the subdirectories on a specific removable disk. You will need to ascertain which are your removable disk drive letters. You obviously will have Drive A but if you have a Zip drive or CD-ROM drive, you will need to know the drive letter for each device. Once the user keys in COMPILE, the REPLY command in the batch file asks which drive the user wants to use to compile a list of files. You will have to use the scan codes in Appendix C to determine which keys the user may press depending on the drive letters on your system. The user may also quit once the list of files is completed. All the information from all the disks that are cataloged is collected in one file called TEMP.TXT.

**D-c** Analyze the following batch file:

```
@ECHO OFF
CLS
ECHO.
ECHO Key in a Drive letter to compile a list of files on Drive A.
ECHO Use Q to quit. Use A for Drive A.
ECHO.
REPLY
IF ERRORLEVEL 65 IF NOT ERRORLEVEL 66 GOTO DRIVEA
IF ERRORLEVEL 81 IF NOT ERRORLEVEL 82 GOTO END
IF ERRORLEVEL 97 IF NOT ERRORLEVEL 98 GOTO DRIVEA
IF ERRORLEVEL 113 IF NOT ERRORLEVEL 114 GOTO END
:DRIVEA
SET DRV=A
GOTO LIST
:LIST
CLS
ECHO You are now compiling a list of all the files on Drive %DRV%
ECHO Listing files for Drive %DRV%
ECHO. >> TEMP.TXT
DIR %DRV%:\ /S /A /ON >> TEMP.TXT
ECHO. >> TEMP.TXT
ECHO You have compiled a list of all the files on Drive %DRV%
ECHO to a file called TEMP.TXT on the default drive.
```

**PAUSE**
**:END**

**D-d** Edit and create a batch file called **COMPILE.BAT** in the **CHAP11** subdirectory based on the above model.

**D-e** Document it with your name and date and explain the purpose of the batch file.

**D-f** The file TEMP.TXT should first be deleted in the batch file.

**D-g** Following REPLY, add more choices for the user. The additional choices will allow the user to choose any removable drive. The user should be able to return to the REPLY line to decide which drive to accumulate more files from. This will give the user the opportunity to either choose a different removable drive, or to choose the same one and place a new disk or CD in it. (*Hint:* Remember, adding choices also means adding more IF ERRORLEVEL.)

**D-h** Use the same REPLY in the batch file to allow the user to view the **TEMP.TXT** file on the screen. Remember, the file will be long and you want the user to be able to view the entire file. When finished viewing, the user should be returned to REPLY to either compile more files or to quit. (*Hint:* Remember, every time you add a choice, you must add more IF ERRORLEVEL statements.)

**D-i** Key in the following: `A:\CHAP11>`**CD ** Enter

**D-j** Use EDIT to modify **NAME.FIL**, changing Problem C to Problem D.

**D-k** Key in the following: `A:\>`**GO NAME.FIL CHAP11\COMPILE.BAT** Enter
Follow the instructions on the screen.

**D-l** In Notepad, click **File**. Click **Print**. Click the **Print** button. Click **File**. Click **Exit**.

## Problem E: Challenge Assignment

The following batch file change is difficult, so do it only if you want a challenge.

**E-a** Make **CHAP11** the default directory.

**E-b** Copy **SWAP.BAT** (created in Problem Set I—Problem C) to a new file called **SWAP2.BAT**.

**E-c** Document it with your name and date and explain the purpose of the batch file.

**E-d** The batch file should perform as follows:

- When you key in SWAP2 *dirname filename1 filename2*, the file names will be reversed in the specified directory. The batch file will change to the specified directory prior to performing the "swap."
- One of the items you wish to test for is the existence of a directory. (*Hint:* Remember %1\Nul.) If you key in a subdirectory name where a file name is expected, the batch file should take you to a message that tells you that you keyed in a directory name, not a file name.

- If you key in only SWAP2, the batch file will take you to a message that tells the user how to use the command correctly.
- Be sure the batch file returns the user to the **CHAP11** subdirectory.

**E-e** Key in the following: `A:\CHAP11>`**CD ** Enter

**E-f** Use Edit to modify **NAME.FIL**, changing **Problem C** to **Problem D**.

**E-g** Key in the following: `A:\>`**GO NAME.FIL \CHAP11\SWAP2.BAT** Enter
Follow the instructions on the screen.

**E-h** In Notepad, click **File**. Click **Print**. Click the **Print** button. Click **File**. Click **Exit**.

### Problem F: Challenge Assignment

The following batch file change is very difficult, so do it only if you want a *real* challenge.

**F-a** In the **Chap11** directory, copy **SWAP2.BAT** to **SWAP3.BAT**.

**F-b** Change **SWAP3.BAT** to allow the user to key in either **SWAP3 dirname file1 file2** or **SWAP3 file1 file2 dirname.**

**F-c** Use Edit to modify **NAME.FIL**, changing **Problem D** to **Problem E**.

**F-d** Key in the following: `A:\>`**GO NAME.FIL \CHAP11\SWAP3.BAT** Enter
Follow the instructions on the screen.

**F-e** In Notepad, click **File**. Click **Print**. Click the **Print** button. Click **File**. Click **Exit**.

## Problem Set III—Brief Essay

1. The following is a batch file called TEST.BAT:

```
@ECHO OFF
:AGAIN
IF \%1\==\\ GOTO END
IF %1==LIFE ECHO LIFE
IF %1==MINE ECHO MINE
IF %1==YOURS ECHO YOURS
IF \%1\==\\ GOTO END
IF NOT EXIST %1 GOTO NEXT
TYPE %1
:NEXT
SHIFT
GOTO AGAIN
:END
```

LIFE, MINE, and YOURS are variables. MY.FIL is a file that is on the disk. Analyze this file and describe what will happen and why when you key in each of the following at the command line:

a. TEST LIFE
b. TEST MINE
c. TEST YOURS
d. TEST MY.FIL

e. **TEST life**
f. **TEST my.fil**

2. The following is a batch file. The batch file is called **COPI.BAT**. The purpose of this batch file is so that the user may copy many different files to any floppy disk that the user specifies. Thus, if the user keyed in COPY A:*.TXT *.NEW, all the **.TXT** files and all the **.NEW** files would be copied to the disk in Drive A. The contents of **COPI.BAT** are:

```
IF "%1"=="" GOTO END
FOR %%a IN (a A) DO IF "%%a"=="%1" GOTO drivea
FOR %%b IN (b B) DO IF "%%b"=="%1" GOTO driveb
:drivea
SHIFT
:newa
IF "%1"=="" GOTO END
ECHO copying %1
COPY %1 A:
SHIFT
GOTO newa
:driveb
SHIFT
:newb
IF "%1"=="" GOTO END
ECHO copying %1
COPY %1 B:
SHIFT
GOTO newb
:END
```

Analyze this file and describe what will happen and why when you key in each of the following at the command line:

a. **COPI**
b. **COPI A C:\WUGXP*.TMP C:\WUGXP*.99**
c. **COPI B C:\WUGXP*.TMP C:\WUGXP*.99**
d. **COPI A:**
e. **COPY B:**

Since most people no longer have a Drive B, how could you modify this batch file so you could use a removable Zip drive with the drive letter of H:?