|  |  |
| --- | --- |
| **Title:** | CMD Shell Scripting Primer Lab 1 |
| Updated: | Jan 8th 2013 |
| Overview: | This lab is a primer for those who may be unfamiliar or rusty with working in the Windows CMD shell environment. |
| Task/Activity: | 1. Working with Command Redirection Operators |
|  | 1. Working with FOR, SET and System Variables |
|  | 1. Working with IF, ELSE and System Variables |
|  | 1. Working with FC, FIND, FINDSTR and Redirection |

1. **Working with Command Redirection Operators**

You can use redirection operators to redirect command input and output streams from the default locations to different locations. The input or output stream location is called a **handle**

|  |  |
| --- | --- |
| Redirection operator | Description |
| **>** | Writes the command output to a file or a device, such as a printer, instead of the Command Prompt window. |
| **<** | Reads the command input from a file, instead of reading input from the keyboard. |
| **>>** | Appends the command output to the end of a file without deleting the information that is already in the file. |
| **>&** | Writes the output from one handle to the input of another handle. |
| **<&** | Reads the input from one handle and writes it to the output of another handle. |
| **|** | Reads the output from one command and writes it to the input of another command. Also known as a pipe. |

Here is a list of the three most common shell handles. Other handles are defined individually by the command shell application.

|  |  |  |
| --- | --- | --- |
| Handle | Numeric equivalent of handle | Description |
| STDIN | 0 | Keyboard input |
| STDOUT | 1 | Output to the Command Prompt window |
| STDERR | 2 | Error output to the Command Prompt window |

1. Begin by opening a cmd shell. Press the **Windows Key** + **R**, type **cmd** and press **Enter**.

|  |
| --- |
| Note: To open an Administrative cmd shell, Press the **Windows Key + R**, type **cmd**, and press **<Ctrl> <Shift> <Enter>** (you may need to press **<Alt> + Y** if prompted with UAC) |

1. Use the **echo** command to **redirect** the text “Hello Earthling” to a file called **hello.txt**.

C:\>**echo Hello Earthling,>hello.txt**

To view the contents of the hello.txt file from the cmd shell, enter the following command:

C:\>**type hello.txt**

|  |
| --- |
| The echo command sends the characters that follow the command to the designated output. The default output is stdout(1) but you can redirect the standard output to a file or device. The type command outputs the contents of the hello.txt file(in ASCII) to STDOUT, the console window. |

1. Use the **echo** command to **append** the text “How YOU doing?” to the **hello.txt** file

C:\>**echo How YOU doing?>>hello.txt**

Again enter:

C:\>**type hello.txt**

1. Using echo, redirect the following text to a file called **file.txt**.

C:\>**echo Zulu >file.txt**

C:\>**echo Bravo >>file.txt**

C:\>**echo Alpha >>file.txt**

C:\>**echo Lima >>file.txt**

1. Execute the sort command and **read in** file.txt.

C:\>**sort < file.txt**

You can use the “&” redirection operator to duplicate input/output from one handle to another (where applicable/enabled). It is often used to capture/redirect STDERR to STDOUT when a shell script is run from a batch file. This is also known as handle duplication.

1. Execute the following command (note: this command assumes the X: drive is nonexistent on your system):

C:\>**dir x:\ >c:\file.txt**

C:\>**type c:\file.txt**

The result of the previous command is: “The system cannot find the path specified.”, and the file.txt file is empty. The error, STDERR (2), was **not** wrote to the file.txt file; by default only STDOUT(1) is the active handle when you redirect command output. Now try the same command with a redirection operator that sends both the STDOUT and STDERR to file.txt.

**STDERR(2)**

C:\>**dir x:\ >c:\file.txt 2>&1 STDOUT(1)**

C:\>**type c:\file.txt**

Often you will need to send the result of one command to the next command. This is done using a vertical line, “|”, also known as the pipe character.

1. Execute the **dir** command, pipe the output to the **sort** command (to sort by column 19, AM/PM) and pipe that output to the **more** command.

C:\>**dir \windows | sort /+19 | more**

1. **Working with FOR, SET and System Variables**
2. Enter the **set** command to view the cmd shell environment variables and their assigned string:

C:\>**set**

When you write scripts you will often want them to be portable between different computer names and to be used by different users. By placing the percent “**%**” characters around an environment variable it will automatically be interpreted as its assigned string variable. Here are some commonly used variables:

|  |  |
| --- | --- |
| Variable Name | Example String Value Assigned to Variable |
| COMPUTERNAME | workstation54 |
| SystemDrive | C: |
| SystemRoot | C:\Windows |
| USERNAME | skippy |
| USERPROFILE | C:\Users\skippy |
|  | |
| **Variable Name** | **Example Discrete Value Variables** |
| CD | The current directory |
| RANDOM | A random number between 0 and 32767 |
| ERRORLEVEL | Set by the previous command |
| DATE | Date format: DOW MM/DD/YYYY |
| TIME | Time format: hh:mm:ss.ms |

Along with using variables by themselves you may often need to trim or extract specific string data from the variable. In the following example you will capture and extract the hh mm ss and ms from the %time% variable, set them as a custom variable, and then use the variable.

1. First let’s look at how delimiting characters separate tokens (strings).

C:\>**echo %time%**

**10:30:53.93** If **:** (colon) and **.** (period) were delimiting characters the four tokens would be:

|  |
| --- |
| **Token1** **Token2**  **Token3** **Token4** |

1. A script or batch file (file extension .cmd or .bat) allows us to run multiple cmd shell commands by executing a single file. We will now create a script that uses the FOR command to set variables by extracting tokens from the current time and assigning them to letter variables (%%a, %%b, etc.). Start by creating a new text file called **timeset.bat** in notepad, enter the following commands, then save the batch file:

C:\>**notepad timeset.bat**

Enter the following text and save the timeset.bat file.

**for /F “tokens=1-4 delims=:.” %%a in (“%time%”) do (**

**set Hour=%%a**

**set Minute=%%b**

*Note: If you set variables from the cmd shell, the single letter replaceable parameters only need one %. When you set parameters in a batch script, you need to use two %% characters. Also, letter parameters are case sensitive.*

**set Second=%%c**

**set Milli=%%d**

**)**

1. Execute the **timeset.bat** file.

C:\>**timeset.bat**

1. Now enter the following command to verify the correct tokens were set to the proper assigned variable:

C:\>**echo The time is %Hour%-%Minute%-%Second%-%Milli%**

1. Now create a new batch file on your own called **dateset.bat** that uses the FOR command to set variables for the %**date%** variable.

*Note: When using the “space” character as a delim, it must be the last specified delim in the delims= statement.*

***WRONG!*** *The “space” character* ***MUST*** *be the last delim before the closing* ***“***

“tokens=1-4 delims= /”

“tokens=1-4 delims=/ ”

***CORRECT!*** *The “space” is the last delim before the closing* ***“***

Write the **dateset.bat** script below:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Working with IF, ELSE, and System Variables**
2. We will now create a script that checks for the existence of the **calc.exe** file--and based on whether it exists or not, echo the appropriate result. Start by creating a new text file in notepad called **checkcalc.bat**.

C:\>**notepad** **checkcalc.bat**

1. Enter the following commands into the **checkcalc.bat** file and save it:

**IF exist %windir%\system32\calc.exe (**

**echo Calculator Found!**

**) ELSE (**

**echo Calculator Not Found!**

**)**

1. Now execute **checkcalc.bat** from the cmd shell.

C:\>**checkcalc.bat**

1. Review the list of common variable names from the previous activity (B) to help you with this one. Create a batch file called **IsAdminLoggedin.bat** and enter following script, replacing the \_\_\_\_\_\_\_\_\_\_\_\_ with the appropriate variable name. Save the batch file and run it to verify functionality.

**If %\_\_\_\_\_\_\_\_\_\_%==Administrator (**

**echo Administrator is currently logged in.**

**) ELSE (**

**echo Warning! %\_\_\_\_\_\_\_\_\_\_% is logged in!**

**)**

After a script or program executes it returns an ERRORLEVEL status to the cmd shell. If the program executes normally/successfully it returns an ERRORLEVEL of “0”, meaning there is no error to report. If something goes wrong during a programs execution, it will often return an error level of “1”, meaning there was an error. Although there are other error levels, you will most often work with 0 and 1.

1. Enter the following command to view a successful error level after a program executes:

C:\>**ping localhost**

C:\>**echo %errorlevel%**

1. Enter the following command to view an error, error level after a program executes:

C:\>**ping somethingthatdoesnotexist**

C:\>**echo %errorlevel%**

1. Create a new batch file called **errorlevelping.bat** and enter the following commands, save it, then run it. The IF command can be used to check the errorlevel and run another command based on the errorlevel status.

C:\>**notepad errorlevelping.bat**

**ping localhost**

**if ERRORLEVEL 0 echo Localhost Ping was a SUCCESS!**

**ping somethingthatdoesnotexist**

**if ERRORLEVEL 1 echo Ping FAILURE!**

1. **Working with FC, FIND, FINDSTR and Redirection**

Comparing files and finding strings of text are often a necessity in many administrative scripts. The windows FC command can be used to compare two different files to show the differences between them. FIND, along with its more modern and capable brother FINDSTR, can be used to search for a text string in a file (or files).

1. Create a text file called **file1.txt** and enter Bob, Jim, Marty and Tammy into the file.

C:\>**notepad file1.txt**

Bob

Jim

Marty

Tammy

1. Now create a text file called **file2.txt** and enter Bob, Skippy, Jim, Marty and Tammy into the file.

C:\>**notepad file2.txt**

Bob

Skippy

Jim

Marty

Tammy

1. Open a cmd shell and execute the following command:

C:\>**fc /L /N file1.txt file2.txt**

1. Now execute the following **find** command to determine if “**Tammy**” is in **file1.txt**:

C:\>**type file1.txt | find /N “Tammy”**

OR

C:\>**find /N “Tammy” file1.txt**

1. Create a text file called **allowednames.txt** and enter the following names in the **allowednames.txt** file:

C:\>**notepad allowednames.txt**

Bob

Jim

Marty

Tammy

1. Execute the following **findstr** command to find names in the **file2.txt** file that are **NOT** listed in the **allowednames.txt** file:

C:\>**findstr /V /G:allowednames.txt file2.txt**

1. Create a **findstr** solution that searches **file2.txt** for all instances of “**Jim**” AND “**Marty**”. Write your command below:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Findstr supports a regular expression search capability (albeit limited). Enter the following command to view the correct syntax for regular expression use:

C:\> **findstr /?**

*Note: The regular expression quick reference is found at the bottom of the help screen. An* ***example*** *of a findstr regular expression looking for an e-mail address in a file called* ***addresses.txt*** *would be as follows:*

*C:\>findstr /R [0-z]\*\@[0-z]\*\.[0-z]\* addresses.txt*

1. Create a file called **ssn.txt**. Enter the following text into the file: **333-3456, 111-22-3344, 12345 Main St.** and save it.

C:\>**notepad ssn.txt**

333-3456

111-22-3344

12345 Main St.

1. Next create a **findstr** regular expression to search for and detect each line that has a social security number. Write your command below:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_