|  |  |
| --- | --- |
| **Title:** | CMD Shell Scripting Primer Lab 2 |
| Updated: | Jan 27th 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. Using CHOICE to return an Index Number |
|  | 1. Comments, clearing the screen, and preventing commands from being displayed |
|  | 1. Passing Arguments/Parameters and SHIFT |
|  | 1. Using GOTO, CALLing a LABEL (Function/Subroutine), and counters with SET |
|  | 1. SETLOCAL and ENDLOCAL |
|  | 1. Using “^” to escape special meaning characters |
|  | 1. Pausing, Waiting and Sleeping (TIMEOUT) |
|  | 1. Using && and || |

1. **Using CHOICE to return an Index Number**

Often in a script file we need to prompt the user to make a selection. Although most selection choices are usually YES/NO, you may need the ability to prompt for additional choices. The **choice** command allows users to select one item from a list of choices and returns the index of the selected choice. The **ERRORLEVEL** variable is set to the index of the key that was selected. Assuming the following choices:

C:\>**choice /C YNC /M “Enter Y for Yes, N for No, and C to Cancel.”**

If the user presses the “C” key, the returned ERRORLEVEL will be “3”.

If the user presses the “Y” key, the returned ERRORLEVEL will be “1”.

If the user presses the “N” key, the returned ERRORLEVEL will be “2”.

Based on what the **ERRORLEVEL** is set to you can use the **IF** command to run the proper response.

1. Create a new batch script called **question.bat**. Enter the following text in the script and run it:

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

**choice /C BFC /M “Enter B for BigMac, F for Fish, and C to Cancel.”**

**if %ERRORLEVEL% EQU 1 echo Thank you for choosing the BigMac.**

**if %ERRORLEVEL% EQU 2 echo Thank you for choosing the Fish.**

**if %ERRORLEVEL% EQU 3 echo I see you decided to cancel.**

C:\>**question.bat**

1. **Comments, clearing the screen, and preventing commands from being displayed**

Often you will want to insert remarks and other comments into your scripts. The two common methods used in legacy batch files and modern cmd shell scripts are the “REM” command, and an alternate technique, using a double colon “::”—a form of fancy label trickery. Because the “::” technique is a bit of a hack, be warned that your “::” remarks will throw errors if you put them inside of ( ) code blocks (unless you know the alternate double-colon and colon-space trick). For this reason, I use “REM” for code comments. Note that not all characters can be successfully remarked out, e.g. %~ will expand as a path operator in a batch file and result in an error regardless of whether you use “REM” or “::”.

1. Create a script in notepad called **myrem1.bat**, enter the following text (see below), save the file, and run it.

C:\>**notepad myrem1.bat**

**REM ##################################**

**REM This command will launch the**

**REM Windows calculator.**

**REM ##################################**

**calc**

C:\>**myrem1.bat**

The remarks are displayed in the shell, but anything with a REM in front of the line is not executed. REM is a great way to troubleshoot scripts as you can comment out the line(s) you suspect are causing problems. To prevent your remarks from being echo’d to the screen use the “echo off” command. To prevent the “echo off” command itself from being echo’d to the screen, put a “@” symbol in front of the command. You may also want to clear the screen to prevent confusion to the user. Use the “cls” command in your script to do this. Edit your **myrem1.bat** file as follows:

Add these two lines to your **myrem1.bat** script.

C:\>**notepad myrem1.bat**

**@ECHO off**

**CLS**

**REM ##################################**

**REM This command will launch the**

**REM Windows calculator.**

**REM ##################################**

**calc**

C:\>**myrem1.bat**

A third technique, typically used for in-line comments, is to create a variable that can’t expand. This can be accomplished by using a “=” after the “%”.

1. Modify your **myrem1.bat** file as follows, then run it:

C:\>**notepad myrem1.bat**

**@ECHO off**

Replace calc with this line:

**CLS**

**REM ##################################**

**REM This command will launch the**

**REM Windows calculator.**

**REM ##################################**

**%= Launch calc % calc & %= After calc is closed, launch Paint % mspaint**

C:\>**myrem1.bat**

1. **Passing Arguments/Parameters and SHIFT**

When a script is run you can natively pass it up to nine arguments. The arguments are numbered 1-9, with 0 being the name of the script itself. See the following example:

C:\>**script.bat bob tom**

|  |  |
| --- | --- |
| **Parameter** | **Value** |
| %0 | script.bat |
| %1 | bob |
| %2 | tom |

1. Create a new script file called **parameter.bat** and enter the following commands in the file:

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

**@ECHO OFF**

**ECHO %0**

**ECHO %1**

**ECHO %2**

**ECHO %3**

**ECHO %4**

**ECHO %5**

**ECHO %6**

**ECHO %7**

**ECHO %8**

**ECHO %9**

1. Now run the **parameter.bat** command with the following arguments:

C:\>**parameter.bat one two three four five six seven eight nine**

1. In order to overcome the nine argument limit of the shell, we can reasonably employ the use of the SHIFT (as in, shift-down) command to change the position of the parameters. Modify the **parameter.bat** as follows:

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

**@ECHO OFF**

**ECHO %0**

**ECHO %1**

**ECHO %2**

**ECHO %3**

**ECHO %4**

**ECHO %5**

**ECHO %6**

**ECHO %7**

**ECHO %8**

**ECHO %9**

**SHIFT /1**

**ECHO %9**

**SHIFT /1**

**ECHO %9**

1. Now run the **parameter.bat** command again, but this time add two additional arguments.

C:\>**parameter.bat one two three four five six seven eight nine ten eleven**

An alternative to handling a large or unknown list of arguments/parameters is to use a **FOR** loop. As you recall, a FOR loop uses letters of the alphabet to handle parameters. You can use a wildcard in the FOR loop to handle large parameter lists.

1. Create a new script file called **forargs.bat** and enter the following commands in the file:

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

**@ECHO OFF**

**FOR %%A in (%\*) DO (**

**ECHO %%A**

**)**

1. Now run the **forargs.bat** script and pass it as many arguments as you want:

C:\>**forargs.bat one two three four five six seven eight nine ten eleven twelve thirteen fourteen fifteen sixteen seventeen eighteen nineteen etc.**

1. **Using GOTO, CALLing a LABEL (Function/Subroutine), and counters with SET**

Legacy DOS command.com scripts required you to call an external script(s) (which would then accept arguments passed from your script—similar to a function or subroutine being passed a parameter in a program). The cmd shell allows us to call within the script using a LABEL. The label is identified by placing a“**:**” character in front of the label name.

1. Create a new script called **going.bat** and enter the following commands:

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

**@ECHO off**

**PING localhost**

**GOTO ERR%errorlevel%**

If the ping is successful and it GOs to the “:ERR0” label we don’t want it to continue on from there—reading the “:ERR1” label too. EOF (end of file) is a cmd shell extension that jumps to the end of the file and exits.

**:ERR0**

**ECHO The Ping was successful**

**GOTO :EOF**

**:ERR1**

**ECHO The Ping was NOT successful**

1. Execute the **going.bat** script. Note the label that was called. *(You can remove the @echo off to see all results of the script.)*

C:\>**going.bat**

1. Now try changing the PING destination from localhost to **0.0.0.5**. Run the script again, it should call the “:ERR1” label this time.

It is often necessary to run a command or subroutine multiple times. The FOR command allows us to loop using a counter that accepts a starting number, a stepping number, and an ending number. But what if you don’t know what the ending number is and you need the script to keep going until all conditions are exhausted? The SET /a command will allow us to perform arithmetic operations. We can place the operation within a LABEL and CALL it.

1. In the directory you are currently writing scripts in, create five empty text files using the TYPE command (which displays contents of a file) and NUL (which serves as an empty string)—redirecting them into the name of the file you want to create. Call the empty text files: **sampleone.txt, sampletwo.txt, samplethree.txt, samplefour.txt** and **samplefive.txt**.

C:\>**TYPE NUL >sampleone.txt**

C:\>**TYPE NUL >sampletwo.txt**

C:\>**TYPE NUL >samplethree.txt**

C:\>**TYPE NUL >samplefour.txt**

C:\>**TYPE NUL >samplefive.txt**

1. Now create a script called **counting.bat** in the same folder you created the sample text files. Enter the following commands in to the script file and run it.

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

%%A will be assigned to each txt file in the FOR loop.

**@ECHO OFF  
FOR %%A in (sample\*.txt) DO CALL :COUNTEM %%A**

**GOTO :EOF  
:COUNTEM  
SET /a count+=1**

My job is to count up by one every time the label is called.

The txt file will then be passed to the :COUNTEM label as an argument

**SET TXTFILE-%count%=%1 && echo TXTFILE-%count%=%1**

%%A becomes %1 when passed as an argument to a label.

1. **SETLOCAL and ENDLOCAL**

SETLOCAL and ENDLOCAL control what are called localized environment changes. The two commands work together to (a) control the shell environment and (b) determine what variables will continue to exist inside of the immediate shell. An implied ENDLOCAL is executed on the completion of all batch files. If you want a variable to exist outside of the cmd shell, you need to modify the registry using **setx** or perform some other administrative configuration.

1. Execute the **counting.bat** script from the previous activity and run it several times. You will notice that the TXTFILE-## variable keeps counting up every time you run the program. You can also type the command SET to view all set variables. When you close the cmd shell, all variables set by “set /a” will be deleted, so often it is not necessary to address variables that are set during a script. However, if you plan on reusing an array of variables, or to call a label with variables more than one time in the duration of a script, it may be necessary to set a localized environment so the variable count will return to the initial value each time it is called.

C:\>**counting.bat** *(run it several times)*

C:\>**set** *(use this command to view the current set variables after running the counting.bat script several times)*

**@ECHO off**

1. Ensure your cmd shell from the previous activity is closed. Now open a new shell and edit the **counting.bat** script in notepad. Insert the **SETLOCAL** command prior to the FOR loop. Execute the script, pressing the UP arrow between each execution, running it several times. You will now notice the TXTFILE-## variable resets at its initial value when the label is called.

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

**@ECHO OFF**

Add SETLOCAL to the script

**SETLOCAL  
FOR %%A in (sample\*.txt) DO CALL :COUNTEM %%A**

**GOTO :EOF  
:COUNTEM  
SET /a count+=1**

**SET TXTFILE-%count%=%1 && echo TXTFILE-%count%=%1**

1. **Using “^” to escape special meaning characters**

Occasionally you may need to use a special/reserved character in a conditional statement, a FOR loop, or to send a special/reserved character to the output stream—instead of having it interpreted by the cmd shell.

1. Try to echo the “|” (pipe) character to a text file:

C:\>**echo | >pipefile.txt**

When the cmd shell interprets the “|” character it tries to pipe the result of the echo command to the next command, resulting in the error: “The syntax of the command is incorrect.“. To escape the special purpose of a character so it is interpreted by the cmd shell as its “character” value, use the “^” (caret) symbol to escape the special character.

1. Try to echo “|” again, but this time place the “^” character in front of it.

C:\>**echo ^| >pipefile.txt**

C:\>**type pipefile.txt** *(to view the contents of the file)*

When storing output and creating reports, HTML and XML are often used to categorize results and they provide an elegant way to view the data. Because HTML/XML use “<” and “>” characters to declare tags, they cannot be sent to stdout or other locations without being interpreted as stdin and stdout themselves. By escaping with the “^” character it is possible to send HTML/XML tags to stdout and other locations.

1. Create a new script file called **myreport.bat** using notepad. Enter the following commands in the script and execute it:

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

**@ECHO OFF**

**ECHO ^<html^> >report.htm**

**ECHO ^<head^> >>report.htm**

**ECHO ^<title^>Report One^</title^> >>report.htm**

**ECHO ^</head^> >>report.htm**

**ECHO. >>report.htm**

**ECHO ^<body^> >>report.htm**

**ECHO ^<h1^>Report Heading H1^</h1^> >>report.htm**

**ECHO ^<hr^> >>report.htm**

**ECHO ^<h2^>Drives on %COMPUTERNAME%^</h2^> >>report.htm**

**ECHO ^<pre^> >>report.htm**

**fsutil fsinfo drives >>report.htm**

**ECHO ^</pre^> >>report.htm**

**ECHO ^<hr^> >>report.htm**

**ECHO ^<h2^>Directory Tree of %USERPROFILE%^</h2^> >>report.htm**

**ECHO ^<pre^> >>report.htm**

**tree /A %USERPROFILE% >>report.htm**

**ECHO ^</pre^> >>report.htm**

**ECHO ^</body^> >>report.htm**

**ECHO ^</html^> >>report.htm**

C:\>**myreport.bat**

1. In this example we will evaluate the results of multiple commands within a FOR loop. The commands will be passing results to the next command within the loop so it will be necessary to escape the special meaning characters. Create a script file in notepad called **escapeinfor.bat,** enter the following commands and execute it:

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

This entire FOR command should be on ONE line

**@ECHO OFF**

**FOR /F "tokens=1-5 delims= " %%A in ('dir %systemdrive%\ ^| findstr /c:"bytes free"') DO ECHO There are %%C %%D available on %systemdrive%\**

C:\>**escapeinfor.bat**

1. **Pausing, Waiting and Sleeping (TIMEOUT)**

Often it is necessary to pause, prompting the user to hit the “go button” again, or to wait for a command to finish before starting the next one, or maybe to sleep for a specific duration of time before continuing with the rest of the script

1. Pausing is an effective method to troubleshoot script files. By forcing the user to press the “any key” to continue, the script writer can disable the @ECHO OFF and view the output of the script results in stdout. Pausing is also an effective way to notify the user of important information so they know what they are getting themselves into before they continue with the script. Create a new script file called **pausing.bat** in notepad, enter the following commands and execute the script:

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

**@ECHO OFF**

**CLS**

**ECHO \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**ECHO ^| ^|**

**ECHO ^| Warning, this script is going to tell you what time it is. ^|**

**ECHO ^| You may find out you are late for dinner. ^|**

**ECHO ^|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_^|**

**ECHO.**

**PAUSE**

**ECHO.**

**ECHO.**

**ECHO The time is %time%**

**ECHO.**

**ECHO.**

**PAUSE**

The START command can be used to launch a separate window to run a specified program or command. One key benefit of using the START command to launch a script is that you can control the title of the cmd shell window, the priority class, and even the processor affinity. You can also require an executed script/program to finish and terminate before the next line of a script file can be processed.

1. Create a new script file called **startwait.bat** in notepad, enter the following commands and execute the script:

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

**START calc**

**START mspaint**

**START write**

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

The START command in the previous script launches each executable nearly simultaneously. To force a program or script to finish and terminate before the next line can be processed enter the /WAIT command before the executable.

1. Modify the startwait.bat file as follows then run the script again:

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

**START /WAIT calc**

**START /WAIT mspaint**

**START /WAIT write**

C:\>**startwait.bat**

Many shell environments support a “sleep” command to suspend program execution for a specified period of time. Unfortunately not all versions of Windows ship with a builtin “sleep” command. Windows 7 ships natively with a TIMEOUT command to suspend execution for a specified time period. If you are on a pre-Vista cmd shell you may have to use tricks like pinging and FOR/IF loops to suspend execution.

1. Create a new script file called **sleeping.bat**. Enter the following commands and execute the script:

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

**@TIMEOUT /T 7 /NOBREAK**

C:\>**sleeping.bat**

1. To prevent an interactive “Waiting for # seconds” and the “CTRL+C” from displaying you can redirect the output to NUL. Modify the sleeping.bat file as follows and execute it again:

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

ADD >NUL

**@TIMEOUT /T 7 /NOBREAK >NUL**

C:\>**sleeping.bat**

1. **Using && and ||**

Usually in a batch script you will want each command to be on a line of its own. This keeps the script clean and easy to read for other network admins. Occasionally you may want to group your commands, running multiple ones on one line, especially if you are running commands interactively from the shell (or if you need to put several commands in one IF statement). You can simply press the up arrow and run multiple commands over and over one line at a time. This is done by using a single “&” (ampersand) character to separate each command, e.g.:

C:\>**ping localhost & dir c:\ & echo Done!**

The drawback to grouping your commands is that the return ERRORLEVEL will be determined by the last command in the set. If the first command fails and the subsequent ones are successful, the ERRORLEVEL returned will be the one set by the last command.

c:\>**ping hostthatdoesnotexist & dir c:\ & echo Done!**

The previous command will return an ERRORLEVEL of “0”, even though the first ping was not successful.

You can also avoid “complex” IF and ERRORLEVEL checking by using && and ||. Placing two “&&” between commands will only allow the next command to run if the preceding command returns an ERRORLEVEL of “0” (success). Additionally, placing two “||” (pipe) characters between commands will only run the next command to run in the preceding command returns an ERRORLEVEL that is NOT “0” (some type of failure).

1. Create a script file in notepad called **multiple.bat**. Enter the following commands in the script and execute it:

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

**ping localhost && dir c:\nul && echo Done!**

**ping fakehost || dir c:\ || echo Finished!**