CSC 368 – Systems Programming Languages
Spring 2017
Homework 1

The following is based on the tutorial at http://www.pcstats.com/articleview.cfm?articleID=1767. There are a few changes, so be consistent with this handout, not the original website. You should use the VM on the cscvc01 cluster. The user ID is Admin, the password is p@ssw0rd with the zero for the letter 'o'.

Beginners Guides: Understanding and Creating Batch Files

Batch files can save time by automating specific actions taken on the computer down to one simple click. They can also potentially hide damaging code, so a good understanding of what they are, how they work, and how to create your own, is crucial to today's IT force. - Version 1.2.0

In this Beginners Guide, PCSTATS is going to walk you through one of the simplest but potentially most powerful ways to customize and simplify the management of your computer: batch files. These text files are easy to create, and only as complex as you want them to be, but they can perform many useful operations from file backups to system configuration quickly and automatically.

At their simplest, batch files are text files which execute one or more command prompt commands in a specific order. The power of a batch file lies in the way that it allows you to combine multiple commands into one batch file 'program' and customize the way that each command operates.

In this article PCSTATS will illustrate what batch files are good for and how to create them. By creating a useful series of batch files designed to allow you to selectively back up files from one location to another, we'll demonstrate the ways that batch files can make your computing life easier while you learn the various option involved in creating them.

What can batch files do for you?

If you read PCSTATS Guide to the Windows Command Prompt, you'll have a passing familiarity with several very useful command prompt commands. Batch files can incorporate any command prompt command (including the switches for that command), execute multiple commands in sequence and choose which commands to use based on user input or the results of previous commands. Essentially, anything you can do in the command prompt you can do better and faster with batch files once you have prepared them.

This article deals with using the Microsoft Windows XP/2K command prompt to create batch files, so commands here may not be 100% compatible with earlier versions of Windows which used a true DOS prompt (and the same holds true in reverse; true DOS batch files may not work under Windows XP). The range of commands available to produce batch files actually decreased with the Windows XP/2K command prompt implementation, but the essential functions are still present.

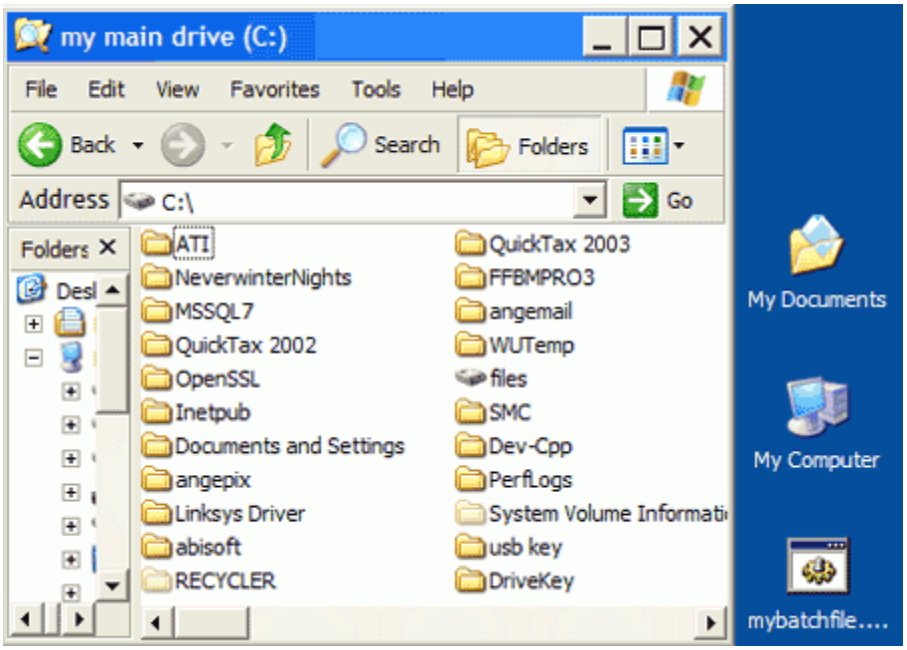
How are batch files created?

As we've said, batch files are simply text files, so all you need to create one is the Windows notepad application. Don't use WordPad or a word processor like Microsoft Word; these do not produce 'pure' text files by default, since they add their own text formatting.

To create a simple batch file, all you need is a single command you want to run, typed into a text file and saved with the .BAT extension, like 'mybatchfile.bat'. Double click this file and it will run your command. Let's test it out...

Creating a BATCH File

First open an explorer window to your c: drive, using Windows Explorer or 'my computer.' Arrange the window so you can see both your desktop and your c: drive contents.



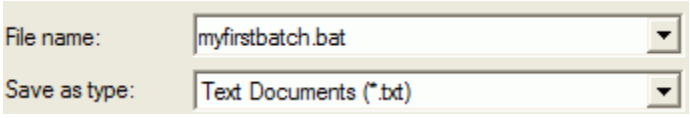
Open the notepad application by going to 'start\all programs\accessories\notepad' or 'start\run' and type 'notepad'. NOTE: You can download Notepad++ to get a language sensitive editor that will use different colors for different language parts.

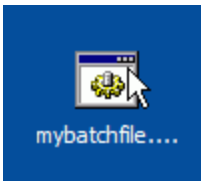
In the blank notepad window, type:

rem <put YOUR NAME here>
md c:\testsource
md c:\testbackup

Replace everything between and including the '<' and the '>' with your name. This creates two directories we will use later.

Now go to 'file' and 'save as'.
(in case you did not read our guide to the command prompt, the 'md'





command instructs the system to create a directory using a name and location following the command.)

Save your first batch file on the desktop as 'myfirstbatch.bat'.

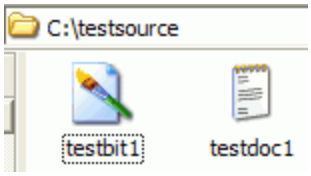
Close notepad and you'll see that 'myfirstbatch.bat' has appeared on the desktop. Double click the file to run it. Check your c: window. The 'testsource' and 'testbackup' directories have appeared. Your first simple batch file is a success! Delete the 'myfirstbatch.bat' file from your desktop.

Preparing for your second batch file

Now we are going to create a useful batch file that will copy all the files in your 'c:\testsource' directory into your 'c:\testbackup' directory each time you run it. We'll also make it so that the next time you run that batch file, it will only copy the files that have changed and new files, not every single one in the 'c:\testsource' directory. Ideally you'd use this batch file to copy your files onto a second hard drive or removable media like a USB key, but we'll leave it as it is for now. Feel free to change the target directory to something more useful later.

First we need to create a couple of dummy files in our c:\testsource directory to give us something to work with:

Create two empty files. Navigate to your c:\testsource directory in Explorer and right click on the empty space inside. Select 'new\text document.' Call your new text file 'testdoc1'. Now right click again and create a new bitmap image. Call your image 'testbit1'. Your c:\testsource directory should now have the following contents:



Creating your second batch file

Now to create a batch file to backup these files into your c:\testbackup directory automatically. Open up notepad and type the following:

rem <YOUR NAME>
@echo off
xcopy c:\testsource c:\testbackup /m /e /y

The '@echo off' line tells the computer not to display anything onscreen when it runs this batch file.

The second line uses the xcopy command to copy all contents of the c:\testsource' directory to c:\testbackup the first time the batch file is run. The second time and all remaining times, it will only copy new files and files which have changed since it was last run. It will not copy unchanged files which it previously copied, even if you delete the copies it made from the c:\testbackup' directory.

Now save your batch file as 'testbackup.bat' on your desktop and double click it to run the script.

Check the contents of your c:\testbackup directory. It should now have copies of the two files you created in c:\testsource. Good stuff. Now open 'testdoc1' in your c:\testsource directory and add some text then save it.

Run your testbackup.bat batch file again, and go to the 'testdoc1' file in the c:\testbackup folder. It should have been updated with the changes you made in the other folder.

You've now created a useful backup utility with a simple two-line batch file that just takes a double click to run. Starting to see the potential usefulness of knowing your batch files yet?

Anatomy of a batch file

As you've seen, batch files are essentially simple programs, using the built in command prompt commands as a programming 'library'.

Each line in a batch file is executed sequentially, and they do not have to be numbered or otherwise identified. The computer simply reads the whole file from top to bottom and performs any commands the batch file contains.

In addition to the standard command prompt commands that can be used in batch files, there are a few additional extras like the '@ECHO off' command we used in the last batch file we created. These can be used to modify how a batch file works. Let's run through some of them with examples of how they work:

Additional Batch file commands

Call
Can be used to call another batch file from within the current one, *for example:*

```
call c:\batchfile2.bat
```

Echo
Used to display information and commands on the screen or prevent them from being displayed.

Echo on
causes all commands in the batch file to be displayed onscreen. This is the default setting.

@Echo off
causes no commands to be displayed. The batch file will run silently unless you use the echo command specifically as below.

echo *what you want on the screen*
Causes whatever text comes after the echo command on the same line to be printed on the screen.f

For
Used to select a specific set of files and run a command on each of them.

for (variable) in (set of files) do (command)

In another example, the batch file line;

```
for %%F in (*.txt) do del "%%F"
```

gives the variable '`%%F`' the value of every file ending in .txt in the current directory, then passes that variable to the DEL command. This means that every file with the .txt extension in the current directory will be deleted. Look here for more on the FOR command.

Goto
Moves to different points within a batch file. The destination point must be indicated with a colon. For example…

goto end
:end
echo this is the end, beautiful friend… the end

If
Performs a command depending on a condition. IF must include an ELSE statement which says what happens if the condition is not met. For example:

if exist c:\myfile.txt (copy c:\myfile.txt d:\myfiles) else echo myfile.txt does not exist

In this example, if the 'myfile.txt' file exists, it will be copied to d:\myfiles. If it does not, a message will be shown indicating this.

Look here for more IF command options.

REM
Rem is used to create comments, or ignored sections in batch files. The computer will ignore any line that begins with REM, so you can use this command to add notations explaining what your file does. For example:

@echo off
rem this batch file is
rem designed to format
rem floppy disks in the a: drive
format a:

Third trial batch file: getting fancy

Now that we've seen some of the extra commands that can be used in batch files, let's play with one of the most powerful of them, the FOR command. In this case, we're going to alter our simple backup batch file and make it a bit more sophisticated. It's going to differentiate between two different types of files (text/Word documents and pictures) and back each file type up to a different directory. To set up for this we need to create two more directories in c:\. Call them

C:\Text
C:\Pics

Delete the existing text and .bmp files in your c:\testsource directory and create a couple of new versions of each.

Now open notepad and enter the following:

```
@echo off
cd c:\testsource
for %%f in (*.doc *.txt) do xcopy c:\testsource\"%%f" c:\text /m /y
for %%f in (*.jpg *.bmp *.gif) do xcopy c:\testsource\"%%f" c:\pics /m /y
```

Now this is a bit more complicated than the files we did before, so let's take a close look at what this batch file is going to do. NOTE: This will be useful for project 1.

cd c:\testsource

Tells the computer that the directory we are going to be working in is c:\testsource

for %%F in (.doc *.txt) do xcopy c:\testsource\"%%F" c:\text /m /y*

This line tells the computer that FOR any file with the .doc or .txt file extension (meaning any standard Word doc or text file), DO an xcopy command to copy that file to the c:\text directory using the same options we used in the last batch file. The confusing looking '%%F' character represents the variable that the FOR command uses to carry out this operation. For example, if your first text file in the c:\testsource directory is 'texttest1.txt', the batch file would look at it, see that it had a .txt extension and assign it as the value of '%%F'. The second part of the command

do xcopy c:\testsource\"%%F" c:\text /m /y

takes whatever %%F is (in this case your 'texttest1.txt' file) and copies it to the c:\text directory. The quotation marks around %%F are to allow the command to deal with file names containing spaces. The command then loops until it has looked at every file in the current directory before moving on to the next part of the batch file.

for %%F in (.jpg *.bmp *.gif) do xcopy c:\testsource\"%%F" c:\pics /m /y*

The only thing that is different here is that we are looking for graphics file extensions instead and copying them to the 'c:\pics' directory.

Save your third batch file on the desktop as 'trickybackup.bat' and try it out. You'll see that your newest creation neatly differentiates between text documents and pictures and splits them up accordingly.

Batch file error levels and the goto command

When a command is executed in a batch file, in addition to any information it may display on the screen, it also returns an error level to the batch file itself. This error level is either errorlevel 0 (meaning no error occurred and the program worked correctly) or errorlevel 1 (an error occurred and that section of the batch file did not work correctly). For example, if you created a simple batch file to format a disk in your floppy (a:) drive like so:

@echo off
format a:

The format command in the batch file would return errorlevel 0 if there was a disk in the drive and it was formatted correctly, while if there was not, or the disk could not be formatted, it would return errorlevel1. By using these errorlevels and the GOTO command, you can set up your batch files to do different things depending on whether the commands in them succeed or fail. For example, you can use the IF command along with the errorlevel returned by the format command to let the user know whether the operation was successful or not:

```
@echo off
format a:
if errorlevel 1 goto error *go to the :error heading if there's an error in format
echo your disk formatted successfully *if format returned no error, show this message
goto end *go to the :end heading to finish the program
:error
echo an error occurred during formatting
:end
echo formatting finished
```

There's no way to cover all the angles of batch files in a single tutorial. You should have picked up enough of the fundamentals and possibilities of batch file creation to start designing your own. Start with modifying the ones we made here to suit your own backup needs, then progress from there. Have fun!

For this homework, turn in the second and third batch files:

testbackup.bat

trickybackup.bat

as attachments to Blackboard.