FORFILES, for your fancier batch file enumeration needs

devblogs.microsoft.com/oldnewthing/20120803-00

August 3, 2012

Raymond Chen

Crack open open the champagne: Batch File Week is finally over!

Variations on the `for /f %%i in ('dir /b ...')` will let you repeat an operation on the contents of a directory, possibly even recursively if you add the `/s` option, with some basic attribute-level filtering if you add the `/a` or /a- flags.

For your fancy recursive file operations, there's a tool called `FORFILES` which iterates through the contents of a directory (recursively if requested), executing a command on each item it finds. It also has additional filtering capability, like selecting files based on their last-modified time. For example, you could copy all files in the current directory which were modified today:

```
forfiles /D +0 /c "cmd /c copy @file \\server\today"
```

Unfortuantely, the `/D` option is not as flexible as one might like. For example, while it can pick files modified today, it can't pick files modified in the last week, because the relative-date-picker knows only how to pick *files modified on or before a date in the past* or *files modified on or after a date in the future*. (Who the heck wants to operate on files modified in the future? Except perhaps the Microsoft Research folks who are working on that time machine.)

You can type `FORFILES /?` for more information on what you can do (and by seeing what's omitted, what you can't do).

If the command you want to execute is rather long, you can offload it back into the batch file being executed:

```
@echo off
if "%1"=="/callback" goto callback
forfiles /D +0 /c "cmd /c call "%~f0" /callback @isdir @file @fsize"
goto :eof
:callback
rem %2 = @isdir
rem %3 = @file
rem %4 = @fsize
if %2==TRUE echo Skipping directory %3.&goto :eof
echo Copying file %3 to \\server\today (%4 bytes)
```

One gotcha here is that since each command runs in a sub-shell, it can read environment variables, but any modifications it makes to environment variables will be lost since the command is modifying only its local environment variables. A workaround for this is to use `FORFILES` to select the data to operate on, but use `FOR` to actually perform the operation. Since `FOR` runs inside the main command interpreter, it can modify environment variables.

```
set TOTALSIZE=0
for /f %%i in ('forfiles /d +0 /c "cmd /c if @isdir==FALSE echo @fsize"') ^
do set /a TOTALSIZE=TOTALSIZE + %%i
```

Here, we use `FORFILES` to enumerate all the files (not directories) modified today and print their sizes. We wrap this inside a `FOR` which reads the sizes and adds them up.

If the operation you want to perform on each file is complex, you can of course offload it into a subroutine call.

```
for /f %%i ^
in ('forfiles /d +0 /c "cmd /c if @isdir==FALSE echo @fsize"') ^
do call :subroutine %%i
```

I'm cheating here because I know that `@fsize` doesn't contain spaces. If you are processing file names, then you need to be more careful.

```
for /f "tokens=*" %%i ^
in ('forfiles /d +0 /c "cmd /c if @isdir==FALSE echo @fname"') ^
do call :subroutine %%i
```

Raymond Chen

Follow