Practical ETL

By

Terry E. Vaughn

2017

Table of Contents

Chapter 1. Tools of the Trade

Chapter 2. File Extraction

In most ETL instances, extraction of data is typically performed on a database object or a data warehouse object. However, there are situations where data extraction from a file is required.

Chapter 3. Database Extraction

Chapter 4. File Content Manipulation

Text file manipulation at the OS command line is one of the more vital skills to possess to make effective use of time when ETL and data analysis tasks are involved. The ability to parse text files of various formats and content and select only that data which is relevant at the moment is a very useful skill to have, particularly if you have a solid command of the tools at hand and you don't have to do a lot of research to achieve the result. Hence, a mastery of the tools necessary for document manipulation improves productivity. Many documents that you will encounter are of a proprietary formatted nature, and therefore, you are forced to use the proprietary tools of the application or program that created the document. Microsoft Word, Excel, Adobe PDF, etc are examples of documents created by proprietary applications. However, text based documents are typically the intermediate documents that you will encounter the most prior to presentation and in general can be manipulated with a variety of tools and are not specific to any application/editor tool. Delimited files such as comma delimited files and fixed format files are examples of ordinary text files. We will explore a few examples of text file manipulation below utilizing various tools available under both Windows and Linux OS environments. We will start with the command line tools available in both Windows (PowerShell) and the Linux (Bash Shell).

Powershell

Example: Customer Orders File Cleanup

We have a text file that contains comma separated data columns with customer order information. We would like to 1.1) extract specific columns of data, 1.2) manipulate the date format of only one of the two date columns, and 1.3) adjust the Unit Price column to always show two decimal places. The final exercise (1.4) will be to aggregate the total net amount of the order and append to the final resultant file. We will use the following file below (file1A.txt) as the initial file to operate on for this example:

File1A.txt

```
line 0: CustCode, PONbr, PODate, DelvDate, Name, PartNbr, QtyOrdered, UnitPrice
line 1: ACME,S4044433,2017-01-01,2017-01-05,ACME Co,101111010,5,4.3
line 2: ACME,S4044433,2017-01-01,2017-01-05,ACME Co,101211010,5,6.12
line 3: CAMLOC,4000001,2017-06-23,2017-06-24,CAMLOC Industries,1001,10,12
line 4: CAMLOC,4000001,2017-06-23,2017-06-24,CAMLOC Industries,2001,20,8.67
line 5: CAMLOC,4000001,2017-06-23,2017-06-24,CAMLOC Industries,3001,20,9.5
line 6: VMP LLC,3239222,2017-08-11,2017-08-13,Victory Machine Products,mypart,5,12.45
```

Exercise 1.1

We first want to reduce the file to 6 columns (custcode, podate, delvdate, partnbr, qtyordered,

unitprice). We will extract these three columns and place in a new file.

```
gc file1A.txt |%{"{0},{2},{3},{5},{6},{7}" -f $_.split(',')}
```

The get-content commandlet aliased as 'gc' writes the content of the file into the pipe command '|' where the for-each-object alias '%' loops through each line of the output and splits the line into fields per the delimiter. The {0},{2},{3},{5},{6},{7} text restricts the content to just columns (custcode, podate, delvdate, partnbr, qtyordered, unitprice).

The resulting file should have the 6 columns we just extracted as shown below:

File1B.txt

```
line 0: CustCode, PODate, DelvDate, PartNbr, QtyOrdered, UnitPrice
line 1: ACME,2017-01-01,2017-01-05,101111010,5,4.3
line 2: ACME,2017-01-01,2017-01-05,101211010,5,6.12
line 3: CAMLOC,2017-06-23,2017-06-24,1001,10,12
line 4: CAMLOC,2017-06-23,2017-06-24,2001,20,8.67
line 5: CAMLOC,2017-06-23,2017-06-24,3001,20,9.5
line 6: VMP LLC,2017-08-11,2017-08-13,mypart,5,12.45
```

Exercise 1.2

Now that the file has been reduced to the required 6 columns, we would like to change the date format of the PODate field from YYYY-MM-DD to YYYY/MM/DD. Although it is easier to use a simple '-replace' cmdlet to change the original character to the target character, this method would change all occurrences of the original character. In this case, we only want to change the format of the PODate field and not the DelvDate field. We will use the [regex] and [string] framework classes and appropriate methods to isolate the PODate field and swap the '-' character with the '/' character for all lines of the file leaving the DelvDate field untouched. The following code achieves this:

```
gc File1B.txt |%{$d=[regex]::split($_,','); $d[1]=$d[1].replace('-','/'); [string]::join(',',$d[0..5])}
```

Using the alias 'gc' once again, the contents of File1B.txt are piped through the '|' command where the for-each-object alias '%' loops through each line of the output and performs three operations (separated by ';') per each line within the foreach loop. The first operation splits the line into fields per the delimiter and assigns them to the $d array using the [regex] .Net Framework class and the static method 'split'. The second operation replaces the element at index '1' of array $d (which is our PODate field), effectively replacing the hyphen '-' with the forward slash character '/'. The third and final operation recreates the original delimited line by using the [string] framework class and static join method [string]::join.

We have piped the above actions into a new file (File1C.txt) with the contents looking like below:

File1C.txt

```
line 0: CustCode, PODate, DelvDate, PartNbr, QtyOrdered, UnitPrice
line 1: ACME,2017/01/01,2017-01-05,101111010,5,4.3
line 2: ACME,2017/01/01,2017-01-05,101211010,5,6.12
line 3: CAMLOC,2017/06/23,2017-06-24,1001,10,12
line 4: CAMLOC,2017/06/23,2017-06-24,2001,20,8.67
line 5: CAMLOC,2017/06/23,2017-06-24,3001,20,9.5
line 6: VMP LLC,2017/08/11,2017-08-13,mypart,5,12.45
```

Exercise 1.3

The third task of this objective is to ensure that the UnitPrice column is always formatted to show two decimal places to the right of the decimal in all occurrences (all lines). You will notice from File1C.txt that the UnitPrice column has various formatted occurrences. For example line 1 has the price at 4.3. We would like this to be displayed as 4.30. Line 3 has a UnitPrice of 12. Again we would like this to be formatted uniformly with "12.00". So, we will once again attempt to target a specific column in the data file and change the formatting structure of this column only. The following code achieves the desired result:

```
gc File1C.txt |%{$d=[regex]::split($_,','); $d[5]="{0:N2}" -f [double]$d[5]; [string]::join(',',$d[0..5])}
```

Using the alias 'gc', the contents of File1C.txt are piped through the '|' command where the for-each-object alias '%' loops through each line of the output and performs three operations (separated by ';') per each line within the foreach loop. The first operation (highlighted in blue) splits the line into fields per the delimiter and assigns them to the $d array using the [regex] .Net Framework class and the static method 'split'. The second operation (highlighted in green) converts the column element at index '5' of array $d (which is our UnitPrice field) from a string to a double, and then formats to numeric with 2 decimal places using the format specifier {0:N2}. The third and final operation (highlighted in red) recreates the original delimited line by using the [string] framework class and static join method [string]::join.

Note: The code statement above will generate an 'informative' error stating that the column header label 'UnitPrice' cannot be converted to double. The error is displayed to standard error output (the screen), however the code still works as intended and will output 'UnitPrice' followed by the correctly formatted numerical values. The error is generated due to line 0 containing the header column names, and the [double] typecast is unable to cast 'UnitPrice' to a double. You can adjust the code to ignore line 0 or you can provide a conditional check for an actual double in the column with the following remake of the above code :

```
gc File1C.txt |%{$d=[regex]::split($_,','); if( [double]::tryparse($d[5],[ref]0.0)) {$d[5]="{0:N2}" -f [double]$d[5]} else {$d[5]=$d[5]}; [string]::join(',',$d[0..5])}
```

in this case, the [double]::tryparse method from the double class is employed for the conditional checking statement. Written long hand with comments, this statement becomes :

```
if ( [double]::tryparse($d[5], [ref]0.0 )   ….if the value of $d[5] is a double
{ $d[5]="{0:N2}" -f [double]$d[5] } ....reformat the value of $d[5] with 2 decimal precision
else
{ $d[5]=$d[5] } ….assign existing string as-is (i.e. the column label in line 0)
```

The [ref] typecast with the 0.0 object is essentially a design necessity of the tryparse method requiring a reference to a double object as the second argument. In effect, it is a useless initialization, but a necessary one none-the-less.

The contents of the above reformatting create the following file (FileD.txt). Note the changes applied to the UnitPrice column.

File1D.txt

line 0: CustCode, PODate, DelvDate, PartNbr, QtyOrdered, UnitPrice
line 1: ACME,2017/01/01,2017-01-05,101111010,5,4.30
line 2: ACME,2017/01/01,2017-01-05,101211010,5,6.12
line 3: CAMLOC,2017/06/23,2017-06-24,1001,10,12.00
line 4: CAMLOC,2017/06/23,2017-06-24,2001,20,8.67
line 5: CAMLOC,2017/06/23,2017-06-24,3001,20,9.50
line 6: VMP LLC,2017/08/11,2017-08-13,mypart,5,12.45

Exercise 1.4

The final task of this example is to determine the total sales amount for the entire file. This will be achieved by multiplying the Quantity and UnitPrice columns together for each row and summing the amount at the bottom of the file. A summary record will be appended to the bottom of the file with the total sales amount in the UnitPrice Column.

```
$x = 0; gc File1D.txt |%{ $d = [regex]::split($_,','); $x += ([double]$d[4] * [double]$d[5]); $tot =
“{0:N2}” -f $x; [string]::join(',',$d[0..5]) }; write-output ",,,,,$tot"
```

The first statement in the above powershell code initializes the variable $x to zero. This variable will be used to accumulate the summed total sales, and it must be set to zero upon each execution of the above code. Otherwise, the variable would continue to accumulate per each execution to the += operator. The contents of File1D.txt is piped to the regex::split method (shown in blue) to separate the columns into the $d array. The statement with the green background typecasts the columns 4 and 5 and of the array and assigns the product of these two columns to the variable $x. This variable is then accumulated with the += operator and is formatted to two decimal places. The original line is then recreated with the [string]::join method and sent to output (highlighted in yellow). Finally, the total is written to standard out in the appropriate column (highlighted in purple) with leading commas to

ensure proper placement in the file (...in the UnitPrice column). The resultant file (File1E.txt) is shown below with the total net sale provided in line 7 for the accumulated orders in the file.

File1E.txt

```
line 0: CustCode, PODate, DelvDate, PartNbr, QtyOrdered, UnitPrice
line 1: ACME,2017/01/01,2017-01-05,101111010,5,4.30
line 2: ACME,2017/01/01,2017-01-05,101211010,5,6.12
line 3: CAMLOC,2017/06/23,2017-06-24,1001,10,12.00
line 4: CAMLOC,2017/06/23,2017-06-24,2001,20,8.67
line 5: CAMLOC,2017/06/23,2017-06-24,3001,20,9.50
line 6: VMP LLC,2017/08/11,2017-08-13,mypart,5,12.45
line 7: ,,,,,597.75
```

Chapter 5. File Format Transformation

Chapter 6. File Loading

Chapter 7. Database Loading

Chapter 8. Data Analysis

Chapter 9. Network Communication

Chapter 10. File and Directory Management

Chapter 11. Archiving

Chapter 12. Appendix 1. Powershell Aliases

Name	Definition
%	Foreach-Object
?	Where-Object
ac	Add-Content
asnp	Add-PSSnapIn
cat	Get-Content
cd	Set-Location
chdir	Set-Location
clc	Clear-Content
clear	Clear-Host
clhy	Clear-History
cli	Clear-Item
clp	Clear-ItemProperty
cls	Clear-Host
clv	Clear-Variable
compare	Compare-Object
copy	Copy-Item
cp	Copy-Item
cpi	Copy-Item
cpp	Copy-ItemProperty
cvpa	Convert-Path
dbp	Disable-PSBreakpoint
del	Remove-Item
diff	Compare-Object
dir	Get-ChildItem
ebp	Enable-PSBreakpoint
echo	Write-Output
epal	Export-Alias
epcsv	Export-Csv
epsn	Export-PSSession
erase	Remove-Item
etsn	Enter-PSSession
exsn	Exit-PSSession
fc	Format-Custom
fl	Format-List
foreach	Foreach-Object

ft	Format-Table
fw	Format-Wide
gal	Get-Alias
gbp	Get-PSBreakpoint
gc	Get-Content
gci	Get-ChildItem
gcm	Get-Command
gcs	Get-PSCallStack
gdr	Get-PSDrive
ghy	Get-History
gi	Get-Item
gjb	Get-Job
gl	Get-Location
gm	Get-Member
gmo	Get-Module
gp	Get-ItemProperty
gps	Get-Process
group	Group-Object
gsn	Get-PSSession
gsnp	Get-PSSnapIn
gsv	Get-Service
gu	Get-Unique
gv	Get-Variable
gwmi	Get-WmiObject
h	Get-History
history	Get-History
icm	Invoke-Command
iex	Invoke-Expression
ih	Invoke-History
ii	Invoke-Item
ipal	Import-Alias
ipcsv	Import-Csv
ipmo	Import-Module
ipsn	Import-PSSession
ise	powershell_ise.exe
iwmi	Invoke-WMIMethod
kill	Stop-Process
lp	Out-Printer
ls	Get-ChildItem
man	help

md	mkdir
measure	Measure-Object
mi	Move-Item
mount	New-PSDrive
move	Move-Item
mp	Move-ItemProperty
mv	Move-Item
nal	New-Alias
ndr	New-PSDrive
ni	New-Item
nmo	New-Module
nsn	New-PSSession
nv	New-Variable
ogv	Out-GridView
oh	Out-Host
popd	Pop-Location
ps	Get-Process
pushd	Push-Location
pwd	Get-Location
r	Invoke-History
rbp	Remove-Breakpoint
rcjb	Receive-Job
rd	Remove-Item
rdr	Remove-PSDrive
ren	Rename-Item
rjb	Remove-Job
rm	Remove-Item
rmdir	Remove-Item
rmo	Remove-Module
rni	Rename-Item
rnp	Rename-ItemProperty
rp	Remove-ItemProperty
rsn	Remove-PSSession
rsnp	Remove-PSSnapin
rv	Remove-Variable
rvpa	Resolve-Path
rwmi	Remove-WMIObject
sajb	Start-Job
sal	Set-Alias
saps	Start-Process

sasv	Start-Service
sbp	Set-PSBreakpoint
sc	Set-Content
select	Select-Object
set	Set-Variable
si	Set-Item
sl	Set-Location
sleep	Start-Sleep
sort	Sort-Object
sp	Set-Property
spcb	Stop-Job
spps	Stop-Process
spsv	Stop-Service
start	Start-Process
sv	Set-Variable
swmi	Set-WMIInstance
tee	Tee-Object
type	Get-Content
where	Where-Object
wjb	Wait-Job
write	Write-Output