

Going Command(o): Power(Shell)ing Through Your Workload

Richann Jean Watson, DataRich Consulting;
Louise S. Hadden, Abt Associates Inc.

ABSTRACT

Simplifying and streamlining workflows is a common goal of most programmers. The most powerful and efficient solutions may require practitioners to step outside of normal operating procedures and outside of their comfort zone. Programmers need to be open to finding new (or old) techniques to achieve efficiency and elegance in their code: SAS® by itself may not provide the best solutions for such challenges as ensuring that batch submits preserve appropriate log and lst files; documenting and archiving projects and folders; and unzipping files programmatically. In order to adhere to such goals as efficiency and portability, there may be times when it is necessary to utilize other resources, especially if colleagues may need to perform these tasks without the use of SAS software. These and other data management tasks may be performed via the use of tools such as command-line interpreters and Windows PowerShell (if available to users), used externally and within SAS software sessions. We will also discuss the use of additional tools, such as WinZip®, used in conjunction with the Windows command-line interpreter.

INTRODUCTION

In the early 1960s, mainframe and minicomputer manufacturers developed “disk operating systems” to control the usage of hard disks and floppy disk drives that allowed access to and storage of sequential and other data on media. At the time, most computers were using tape drives, or had no storage device at all. As computers evolved, the concept of a disk operating system (DOS) became more well known, and different manufacturers developed their own versions, often with “DOS” in the name. In the late 70s and 80s, most home and personal computers were using some form of DOS. The original Mac had its entire operating system on 3 ½ “floppy” drives, with disks being swapped out for operating commands and file storage. Eventually, versions of DOS, once loaded from disk, were installed in ROM, and then in computer chips. The Windows, UNIX, and Linux platforms evolved, usually incorporating, and assimilating various flavors of the original DOS versions.

The purpose of this paper is to explore the underpinnings and remnants of original DOS versions in the Windows operating system that still have great utility today and can be used to efficiently perform operations that would be very difficult, or time consuming as multiple manual point and click operations in Windows. While we are focusing on the Windows platform in this paper, many of the operations we describe can also be performed on UNIX and Linux systems via shells such as the Korn shell and BASH.

The two primary tools used in this exploration are the COMMAND-LINE INTERPRETER (CLI), sometimes referred to as CMD, and PowerShell. Several use cases will be described for each, using system prompts and utilizing these operating system tools and other command line utilities within instances of SAS via pipes and the X command. Additionally, SAS runs on command prompts behind the scenes, allowing users to add and modify parameters contained in the sasv9.cfg file on the fly, and even incorporating an external configuration file. Examples with code snippets are referred to in the text of the paper, and full sample code are provided on a Github page (<https://github.com/rwatson724/CMD-PowerShell-Scripts>). We encourage readers to check the many references in the paper and documented in the references section for more detail.

THE BASICS

With all software tools, there are hundreds of commands and diverse syntax. A high-level overview of each tool is provided, while only the commands to illustrate the examples are detailed. Links to various sites that have a listing of the commands, and their syntax and options, are provided for future reference.

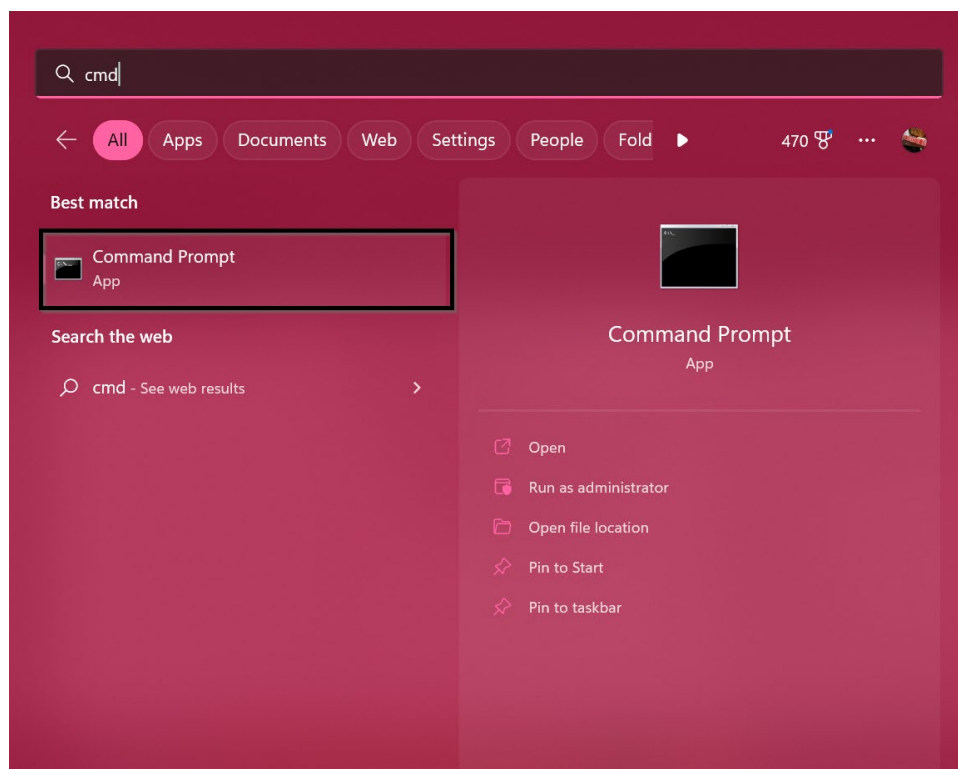
COMMAND-LINE INTERPRETER (CLI) PROMPT

The command-line interpreter (CLI) allows you to type commands at a system prompt. In early versions of Windows and MS-DOS the CLI was commonly referred to as COMMAND.COM, but with later versions of Windows it is referred to as CMD.EXE (CMD). (Computer Hope, n.d.)

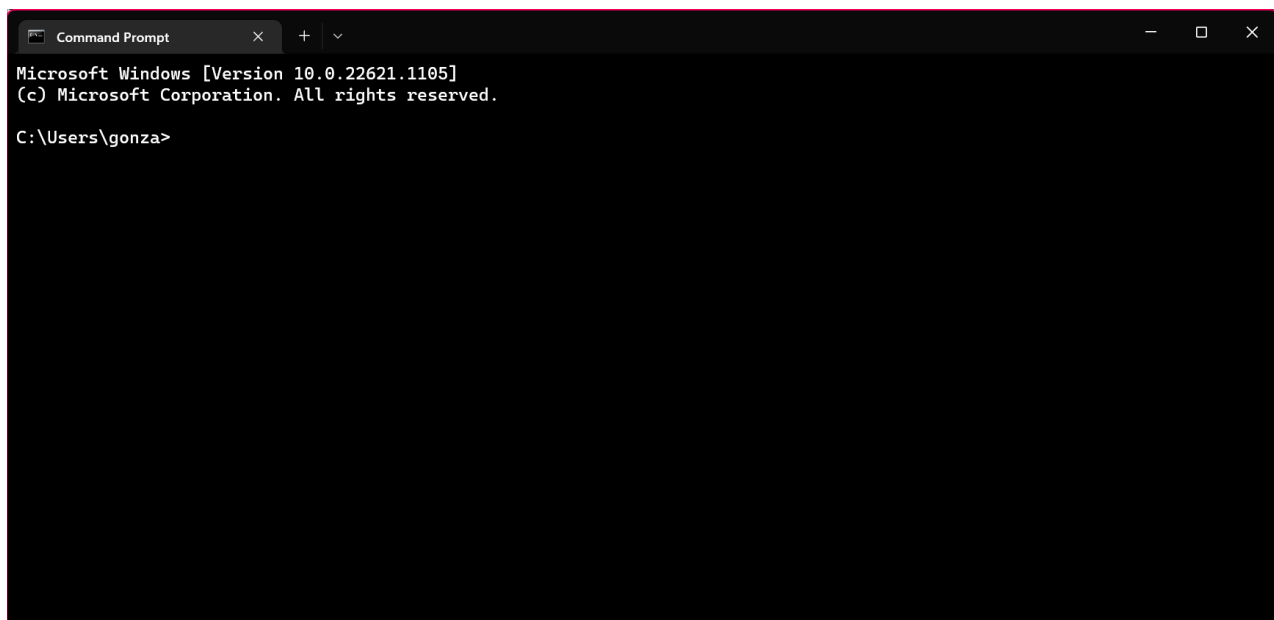
With the use of the CLI, you can perform tasks that are generally performed in a windowing environment with a point and click of the mouse. With shell commands in the CLI, you can create batch files or custom scripts that will allow you to automate many of these tasks or access the CLI within your SAS programs.

Command Prompt (CMD)

In Windows, to access the command-line interpreter, you can enter CMD on the search bar to find the Command Prompt app as shown in Screenshot 1. Once the Command Prompt is found, you can click on the app to open the command-line interpreter (Screenshot 2). The authors highly recommend that you pin your system's CLI to your task bar for further explorations and uses.



Screenshot 1: Searching for Command-Line Interpreter



Screenshot 2: Command-Line Interpreter via CMD

CMD Basics - Variables Versus Parameters

In order to understand some of the commands, we first must understand the difference between a variable and a parameter. Within CMD there are environment variables. These variables are only available during the current CMD session and can be referenced in the current CMD session, by enclosing it within % signs. For example, in Program 1, **SAS**, **PTH**, **PLOG** and **PLST** are environment variables. They are referenced in the last line of the batch script in Program 1 by putting a % on each side of the variable. There are some default environment variables, such as CD (current directory), DATE (current date), TIME (current time). Note that the format of the date and time will vary based on the system (Sheppard, How-to: Windows Environment Variables, n.d.).

Parameters also known as arguments are passed in a batch script (Sheppard, How-to: Pass Command Line arguments (Parameters) to a Windows batch file., n.d.). The value of a parameter can be retrieved by using its position on the command line preceded by a %. For example, the Program 1 script has two arguments that are passed: 1. directory of the program; 2. name of the program. In the batch script (Program 1), the name of the program is referenced as %2 because it is the second argument passed.

```
set sas="C:\Program Files\SASHome\SASFoundation\9.4\sas.exe" -CONFIG "C:\Program
Files\SASHome\SASFoundation\9.4\nls\en\sasv9.cfg"

set pth=-sysin "C:\Users\gonza\Desktop\Conferences\Drafts\Commando\%1\programs\%2.sas"
set plog=-log "C:\Users\gonza\Desktop\Conferences\Drafts\Commando\%1\Logs\"
set plst=-print "C:\Users\gonza\Desktop\Conferences\Drafts\Commando\%1\Lsts\"
%sas% %pth% %plog% %plst%
```

Program 1: Sample of a Batch Script – SINGLE.BAT

Within a SAS session, Program 1 can be executed with an X command and the two parameters specified, "Batch" and "PRXMATCH". These parameters are passed to the script and used for %1 and %2 respectively.

```
x "C:\Users\gonza\Desktop\Conferences\Drafts\Commando\SINGLE.bat Batch PRXMATCH";
```

In some commands within a batch file, a parameter will need to be referenced with two %'s (Sheppard, How-to: Pass Command Line arguments (Parameters) to a Windows batch file., n.d.), which will be illustrated in the examples below.

CMD Commands and Examples

There are a number of commands that can be used within CMD with each command having its own set of arguments and options. The following are a list of some of the commands that are illustrated in this paper. The commands used in the paper are at a high level and a brief description of how they work are provided. For a full list of Windows CMD commands, options and extensive details visit SS64.com (Sheppard, An A-Z Index of Windows CMD commands., n.d.).

SET

The first command used is the SET command. This assigns values to environment variables. The SET command can assign a string or use an expression.

SET	Syntax	Description
Creating, removing or displaying an environment variable. The variable is only available during the current CMD session and the variable name itself is not case sensitive but the value can be. Spacing is NOT ignored.	SET <i>variable</i>	<i>variable</i> new or existing environment variable name
	SET <i>variable</i> = <i>string</i>	<i>string</i> text string to assign to the variable
	SET " <i>variable</i> = <i>string</i> "	<i>expression</i> arithmetic expression
	SET " <i>variable</i> ="	/A see documentation for details on arithmetic expressions
	SET /A " <i>variable</i> = <i>expression</i> "	/P prompt for user input
	SET /P <i>variable</i> =[<i>promptString</i>] SET "	To delete a variable use SET "variable=". This will ensure there is no trailing space after the equal sign. You can also use (SET variable=).

Sheppard op. cit

The expression can utilize other environment variables as seen in the example below. DY, MO and YR use the environment variable **date** and parses the necessary string to assign the value. Note that **date** is a default environment variable, and the format of the date will vary based on the system. For this example, date is stored as "Day MM/DD/YYYY", where Day is the three-character abbreviation for the day of the week, MM represents the two-digit month, DD is the two-digit day and YYYY is the four-digit year. When parsing the string, it starts at 0 and not 1, therefore, while it may seem that month would start in the 5th position, it is actually the 4th position since Day starts in the 0th position.

```
C:\Users\gonza>echo %date%
Thu 02/09/2023
```

Display 1: DATE Environment Variable

If the following code is run on Thursday, Feb 9, 2023, then **date** = 'Thu 02/08/2023' (Display 1), then it should yield the individual components. These individual components can then be combined to create a new date in the desired format (Display 2).

set dy=% date :~7,2%	→ 09
set mo=% date :~4,2%	→ 02
set yr=% date :~10,4%	→ 2023
set today=%yr%-%mo%-%dy%	→ 2023-02-09
set runday=%today%	→ 2023-02-09

```

C:\Users\gonza>set dy=%date:~7,2%

C:\Users\gonza>set mo=%date:~4,2%

C:\Users\gonza>set yr=%date:~10,4%

C:\Users\gonza>set today=%yr%-%mo%-%dy%

C:\Users\gonza>set runday=%today%

C:\Users\gonza>echo DAY=%dy% MON=%mo% YEAR=%yr% TODAY=%today% RUNDAY=%runday%
DAY=09 MON=02 YEAR=2023 TODAY=2023-02-09 RUNDAY=2023-02-09

```

Display 2: Extracting Date Components from DATE Environment Variable

DIR

The DIR command lists all the files and subfolders in the indicated folder. Depending on the options used, we can filter for specific files.

DIR	Syntax	Description																																				
Displays a list of all files and folders in the indicated drive or folder.	DIR [pathname(s)] [display_format] [file_attributes] [sorted] [time] [options]	Pathname can be the drive, folder or files that are to be displayed and can use wildcards. Wildcards: <table><tr><td>*</td><td>match on any characters</td></tr><tr><td>?</td><td>match on any ONE character</td></tr></table> [file_attributes] options: <table><tr><td>/A</td><td>All Files</td><td></td><td></td></tr><tr><td>/A:D</td><td>Folder</td><td>/A:-D</td><td>NOT Folder</td></tr><tr><td>/A:R</td><td>Read-only</td><td>/A:-R</td><td>NOT Read-only</td></tr><tr><td>/A:H</td><td>Hidden</td><td>/A:-H</td><td>NOT Hidden</td></tr></table> [sorted] options: <table><tr><td>/O:N or /O:-N</td><td>Name</td></tr><tr><td>/O:D or /O:-D</td><td>Date & time</td></tr></table> [time] options: <table><tr><td>/T:C</td><td>Creation</td></tr><tr><td>/T:A</td><td>Last Access</td></tr><tr><td>/T:W</td><td>Last Written (default)</td></tr></table> Other available options: <table><tr><td>/B</td><td>Bare format (no heading, file sizes or summary)</td></tr><tr><td>/S</td><td>Include subfolders</td></tr><tr><td>/4</td><td>Display four-digit year</td></tr></table>	*	match on any characters	?	match on any ONE character	/A	All Files			/A:D	Folder	/A:-D	NOT Folder	/A:R	Read-only	/A:-R	NOT Read-only	/A:H	Hidden	/A:-H	NOT Hidden	/O:N or /O:-N	Name	/O:D or /O:-D	Date & time	/T:C	Creation	/T:A	Last Access	/T:W	Last Written (default)	/B	Bare format (no heading, file sizes or summary)	/S	Include subfolders	/4	Display four-digit year
*	match on any characters																																					
?	match on any ONE character																																					
/A	All Files																																					
/A:D	Folder	/A:-D	NOT Folder																																			
/A:R	Read-only	/A:-R	NOT Read-only																																			
/A:H	Hidden	/A:-H	NOT Hidden																																			
/O:N or /O:-N	Name																																					
/O:D or /O:-D	Date & time																																					
/T:C	Creation																																					
/T:A	Last Access																																					
/T:W	Last Written (default)																																					
/B	Bare format (no heading, file sizes or summary)																																					
/S	Include subfolders																																					
/4	Display four-digit year																																					

For example, if we want to only list the files and not the subfolders within the current directory, the option “A:-D” is used. “A” indicates all files which by default includes the subfolders and “-D” indicates exclude the subfolders. In addition, we can look for files with specific naming convention and/or specific extensions. In the example below **rcvd** (environment variable that contains the folder) is scanned and a zip file that has a name that contains the value that is captured in **today** is searched. The option /O:-D indicates that the files are to be sorted by descending date and time. Therefore, if there are more than one file found, it would retrieve the file with the latest date and time. If the file is found the option /B indicates that only the file name is to be returned, that is the metadata associated with the files is not returned. In the example shown in Display 3, even though there are two zip files found with the date of ‘2023-02-09’, the ‘For SAS – 2023-02-09.zip’ had a later creation and therefore is listed first due to the option /O:-D.

```
DIR "%rcvd%\*%today%*.zip" /B /O:-D
```

```
C:\Users\gonza\Desktop\Conferences\Drafts\Commando>dir "%rcvd%\*%today%.zip" /b /o:-d
For SAS - 2023-02-09.zip
ddd2ed_code 2023-02-09.zip
```

Display 3: Retrieving Files with a Specific Naming Convention Using DIR Command

MKDIR (MD)

In addition, to listing out the contents in a directory, we can create a new directory.

MKDIR (MD)	Syntax	Description
Make a new directory or folder. Either MD and MKDIR can be used. Both will make a new directory or folder.	MD [drive:]path [[drive:]path ...]	<i>path</i> directory or folder to be created Spaces (including tabs), commas and semicolons can be used in the folder name; however, they must be enclosed in quotation marks. Multiple directories can be created on the same command line.

Although the example, below shows two separate statements to make two new directories within the current directory. This could have been achieved by specifying both directories on the same command line (e.g., `mkdir __TEMP %today%`). Note that to make a directory, the commands MKDIR and MD can be used interchangeably.

```
mkdir __TEMP
mkdir %today%
```

RMDIR (RD)

If we can create new directories, then it makes sense that we can also remove directories. However, with removing directories there are options that can be incorporated to indicate if the entire folder tree (i.e., subfolders) are to be deleted and if you don't need confirmation to delete. To remove the entire tree the /S option is used and to remove the folder without asking for confirmation uses the /Q option.

RMDIR (RD)	Syntax	Description
Remove or delete a directory or folder. Either RD and RMDIR can be used. Both will make a remove or delete directory or folder.	RD <i>pathname</i> RD /S <i>pathname</i> RD /S /Q <i>pathname</i>	<i>pathname</i> directory or folder to be removed or deleted Available options: /S remove an entire folder tree will delete main folder as well as all files and subfolders /Q quiet does not ask for confirmation of deletion Long pathnames should be place in double quotes. Note that without the /S option, only empty directories will be deleted.

In the example below the temporary folder __TEMP is removed in the current directory, **CD**. **CD** is default environment. Similar to making a directory, there are two commands that can be used to remove a directory, RMDIR and RD.

```
rmdir /s /q "%CD%\__TEMP\"
```

DEL

DEL allows us to specify one or more files to delete. The use of wildcards can be used with part of a filename to identify file(s). There are additional options that be used when determining what deletions are allowed. For example, /S indicates that subfolders should be searched as well, and the file(s) deleted from the subfolders. In addition, /P option allows us to indicate that a Yes/No prompt is needed before deleting. By default, the file(s) will be deleted without a prompt. Note that ERASE command can also be used instead of DEL.

DEL	Syntax	Description
Delete file(s) that are specified.	DEL [options] [file_attributes] files_to_delete	files_to_delete file(s) that are to be deleted can use wildcards Wildcards: * match on any characters ? match on any ONE character Available options: /P Yes/No Prompt before deleting /F Ignore read-only setting and delete anyway (FORCE) /S Delete from all Subfolders (DELTREE) /Q Quiet mode, do not give a Yes/No Prompt before deleting. [file_attributes] options: /A:R Read-only /A:-R NOT Read-only /A:H Hidden /A:-H NOT Hidden

In this example, the file __TEMPTEXT.TXT is deleted.

```
del __TEMPTEXT.TXT
```

MOVE

MOVE allows us to move one or more files from one location to another location. With the MOVE command we can choose to have a prompt asking to confirm the overwriting of files when the source is moved to the target and the target already has a file with the same name. If executing the MOVE command within a batch script, the default is to overwrite without the prompt. If running at the command line, the default is to prompt for an overwrite. Within the MOVE command wildcards are allowed on the source argument but not on the target.

MOVE	Syntax	Description
Move file(s) from one location to another location	MOVE [options] [source] [target]	source path and filename of the file(s) to move can include wildcards target path and filename to move file(s) to Wildcards: * match on any characters ? match on any ONE character Available options: /Y suppress confirmation prompt, when overwriting files /Y enable confirmation prompt, when overwriting files

In this example parameter A is moved to a subfolder which resides in the current directory (CD). The name of the subfolder is contained within the environment variable, **today**.

```
move "%A" "%CD%\%today%"
```

ECHO

To display a message to the screen, we use the ECHO command, ECHO ON. To prevent the messages from displaying to the screen we can turn off the ECHO command with ECHO OFF. By default ECHO is on. The ECHO command can be used to display specific messages or to display help information. When using the ECHO command, the message does not always have to be displayed to the screen. It can be redirected to an external file by using '>' after the message and specifying the name of the file after '>'.

ECHO	Syntax	Description
ECHO by default is turned on. ECHO displays a message to the screen.	ECHO [ON OFF] ECHO [message] ECHO /?	ON display each line of the batch script on screen OFF only display the command output on screen message display a string of characters on the screen ? display help The use of the @ symbol at the start of a line in a batch file is equivalent to using ECHO OFF for the current line. Note that the message can be redirected to an external file if necessary by using the following syntax ECHO [message] > Filepath\Filename If the Filepath is not specified, then it will write to the current directory.

For example, the message below "ZIP file is %rcvdfilename%" is written to an external file that is located in the current directory (CD) and named unzip_%runday%.txt. Both rcvdfilename and runday are environment variables that contain the name of the file and the run date.

```
echo ZIP file is %rcvdfilename% > "%CD%\unzip_%runday%.txt"
```

GOTO

With the GOTO command, we can 'jump' to a location that has a specific label within the batch script. The GOTO command uses the label to indicate where we need to move to. If the GOTO command's label is :EOF, this indicates we are to exit the current subroutine or script. Note that it is possible to have a label of EOF (i.e., there is no preceding colon) and use that on a GOTO command. If the GOTO command indicates that we are to jump to the line labelled EOF, it should be specified as GOTO EOF (i.e., without the colon preceding EOF). This would behave differently than the label :EOF. In other words, GOTO EOF moves to the location in the script that is labelled EOF, where GOTO :EOF will exit. Although it is allowed to have EOF as a label, this can be easily confused with :EOF, thus it is best to avoid using EOF as a GOTO label.

GOTO	Syntax	Description
Direct batch program to move to the line that has the indicated label.	GOTO label GOTO :eof	label predefined label in the batch program label must be defined on a line by itself label must be preceded with a colon and end with a colon, space or a carriage return/line feed :eof predefined label that will exit the current subroutine or script Note that GOTO EOF and GOTO :EOF are not the same. It is possible to create a label EOF (i.e., without the colon on the GOTO statement). However, this could cause confusion so it is best to avoid it.

We can use conditional logic to determine where we need to go to next in the script. In this example, if the condition is true, then we are to jump to a line name FoundFile. Further, in the script will be a line with the label :FoundFile. Thus, if rcvdfilename is set to a value (i.e., not null), then we move to the :FoundFile location in the script.

```
if defined rcvdfilename (goto FoundFile)
...
:FoundFile
```

CALL

With the CALL command, we can execute a specific subroutine within the current batch script or call an external batch script.

CALL	Syntax	Description
Call a batch script or subroutine within another	CALL [drive:][path]filename	pathname fullpath and name of another batch script to run
	[parameters]	parameters command-line argument
	CALL :label [parameters]	:label jump to a labelled line in the current batch script

In this section, CALL is being used to reference a subroutine named **UnZipFile**. The CALL routine specified two arguments for the **UnZipFile** subroutine. The first argument indicates the extraction location which is a subfolder named __TEMP within the current directory (CD). The second argument is the full path and name of the file to be unzipped. The path is stored in the environment variable, **rcvd**, and the name of the file is stored in the environment variable, **rcvdfile**.

```
call :UnZipFile "%CD%\__TEMP\" "%rcvd%\%rcvdfile%"
...
:UnZipFile <ExtractTo> <newzipfile>
...
```

IF/IF-ELSE

Conditional logic can help us determine which portions of the script need to be executed. A condition is checked to see if it is true and if it is the indicated command is executed. Otherwise, if IF-ELSE is used, the else command is executed. If multiple commands need to be executed based on the condition, then the commands need to be enclosed within parenthesis.

IF / IF-ELSE	Syntax	Description
If the condition is met, then it executes the indicated <i>command</i> . If using IF-ELSE, then if condition is not met, then the ELSE <i>command</i> is executed.	IF [NOT] EXIST filename <i>command</i>	Checks to see if a file exists (or does NOT exist).
	IF [NOT] EXIST filename (<i>command</i>) ELSE (<i>command</i>)	Note that a wildcard pattern can be used for the filename
If <i>command</i> contains multiple commands, then they are to be embedded in parenthesis. The use of parenthesis allows for more complex logic.	IF [/I] [NOT] <i>item1</i> == <i>item2</i> <i>command</i> IF [/I] [NOT] " <i>item1</i> " == " <i>item2</i> " <i>command</i>	Compares a text string or environment variable to another text string or environment variable.
	IF [/I] <i>item1</i> <i>compare-op</i> <i>item2</i> <i>command</i>	String comparisons: /I : indicates that the comparison is case insensitive == : two strings are equal
	IF [/I] <i>item1</i> <i>compare-op</i> <i>item2</i> (<i>command</i>) ELSE (<i>command</i>)	Numeric comparisons: EQU : Equal NEQ : Not equal LSS : Less than (<) LEQ : Less than or Equal (≤) GTR : Greater than (>) GEQ : Greater than or equal (≥) Note that < and > cannot be used for numeric comparisons, since these symbols are redirection operators.
	IF [NOT] DEFINED variable <i>command</i>	Checks to see if a variable is null or if it exists.

Using the same example, we use to illustrate GOTO, we now expand this to indicate an ELSE condition. If we do not find a file, instead of going to the labelled line **FoundFile**, we go to the labelled line **NoFile**.

```
if defined rcvdfile (goto FoundFile) else goto NoFile
```

FOR

FOR is similar to a do loop in that it executes a command each time the specified condition is met. As mentioned previously when discussing the difference between environment variables and parameters, we indicated that some commands require the use of two % when referencing the parameter. The FOR command when used within a batch script requires the use of two %.

FOR	Syntax	Description
Perform a <i>command</i> multiple times if the condition is met.	FOR %% <i>parameter</i> IN (set) DO <i>command</i>	%% <i>parameter</i> is set to a value for each iteration of the FOR loop that is executed on several files.
If <i>command</i> contains multiple commands, then they are to be embedded in parenthesis. The use of parenthesis allows for more complex logic.	FOR /F [" <i>options</i> "] %% <i>parameter</i> IN (<i>filenameset</i>) DO <i>command</i> FOR /F [" <i>options</i> "] %% <i>parameter</i> IN ("Text string to process") DO <i>command</i>	%% <i>parameter</i> is set to a value for each iteration of the FOR loop that is executed on a set of files. <i>Filenameset</i> is a one or more files. For a <i>filenameset</i> it parses the contents of the file one line at a time. The contents in the file are broken up into tokens. Available options: delims=xxx delimiter character(s) default for strings is a space or TAB tokens=n indicates which numbered items to read from each line default is 1 multiple tokens are separated by a comma
Note that the first parameter is defined using a single character that is case sensitive and is replaceable. If FOR loop is used in a batch program, then the parameter is defined with two percent signs. If the FOR loop is used at the command line, then only one percent sign is used.	FOR /R [<i>drive:</i>] <i>path</i>] %% <i>parameter</i> IN (set) DO <i>command</i>	FOR loop that traverses down the sub-directories starting at the [<i>drive:</i>] <i>path</i> looking for a match in the set. [<i>drive:</i>] <i>path</i> is the directory where the expected files are located. Set is one or more files specified with wildcards enclosed in parenthesis. Note that if [<i>drive:</i>] <i>path</i> is not specified then the current directory is used.

Here we illustrate two examples of the FOR command. The first one indicates that the FOR loop is to be executed for a set of files (/F). FOR command searches the directory specified by the environment variable, **rcvd**, for a zip file. Note that as the FOR command loops through each file the file name is passed to parameter **I**. If the condition is met (i.e., the file is a zip file), then the commands are executed. In this case, the files are scanned in descending date time order (O:-D) using the creation date (T:C) and subfolders are excluded (A:-D) and we are only concerned about the file name, so we retrieve the 'bare format' (/B) (i.e., we don't need the date, file size, etc.).

```
for /F "delims=|" %%I in ('DIR "%rcvd%\*.zip" /A:-D /B /O:-D /T:C') DO (  
set rcvdfile=%%I  
goto FMTDate  
)
```

The second example of the FOR command utilizes the /R option. This indicates that all folders and subfolders at the indicated location (%**CD**%__TEMP\) are to be searched for SAS7BDAT files. As each file within the current location and within each folder/subfolder is scanned, the file name is passed to the parameter, **A**. If a SAS7BDAT file is found then the file is moved to a folder within the current directory that has a name that is indicated by the environment variable, **today**.

```
for /R "%CD%\__TEMP\" %%A in (*.sas7bdat) do move "%A" "%CD%\%today%\"
```

FORFILES

The FORFILES executes a set of commands for each file selected. Unlike the FOR command, which is a looping command and only executes the command(s) if the condition is met. FORFILES uses the current location by default, or a directory can be specified. To select certain files to perform the command(s) on, the search criteria is preceded by /m. Each command that is to be performed on the file is identified with /c.

FORFILES	Syntax	Description
Perform a command for the indicated file(s) selected. The command should be wrapped in double quotes.	FORFILES [/p <i>path</i>] [/m <i>srchmask</i>] [/s] [/c <i>command</i>]	/p <i>path</i> path where expected file is located default = current folder /m <i>srchmask</i> criteria to select the file /c <i>command</i> command to be executed for each file selected default = "cmd /c echo @file" Additional Command Variables that can be used in the command string: @file name of the file. @fname file name without extension @fdate last modified date of the file @ftime last modified time of the file

In the snippet of code below, we are searching through the folder that is indicated by the environment variable, **rcvd**, for a file that matches the selection criterion specified. /M indicates that we are searching for a file that has the name that is captured in the **__rcvdfile** environment variable. /C indicates we are issuing a command to write the file date to temporary file.

```
forfiles /P "%rcvd%" /M %__rcvdfile% /C "cmd /c echo @fdate" > __TEMPTEXT.TXT
```

Note that FORFILES only retrieves modification date. It does not retrieve creation date.

EXIT

To close the current CMD session or the current batch script, the EXIT command is used. Within a batch script, the option /B can be used to exit a subroutine without ending the entire CMD session.

EXIT	Syntax	Description
Closes the CMD session or can be used to close the current batch script or the exit the current subroutine.	EXIT [/B] [exitCode]	/B exits the script or subroutine when used within a batch script and does not close the CMD session if used on the command-line within CMD, then it closes the CMD session

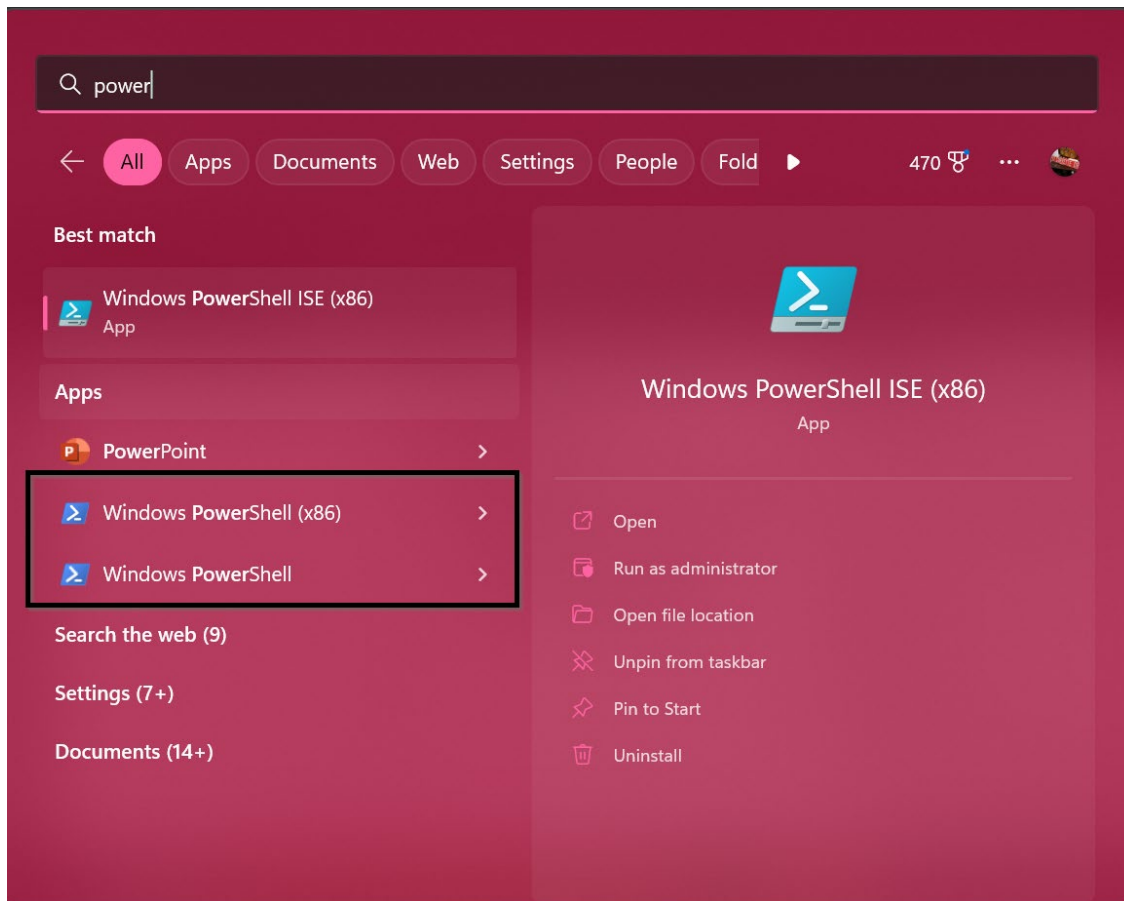
Using the same example above, once the subroutine has been executed, the message indicated on the ECHO command is written to an external file and this portion of the script is exited.

```
call :UnZipFile "%CD%\__TEMP\" "%rcvd%\%rcvdfile%"
echo - File %rcvd%\%rcvdfile% has been unzipped >> "%CD%\unzip_%runday%.txt"
exit /b
```

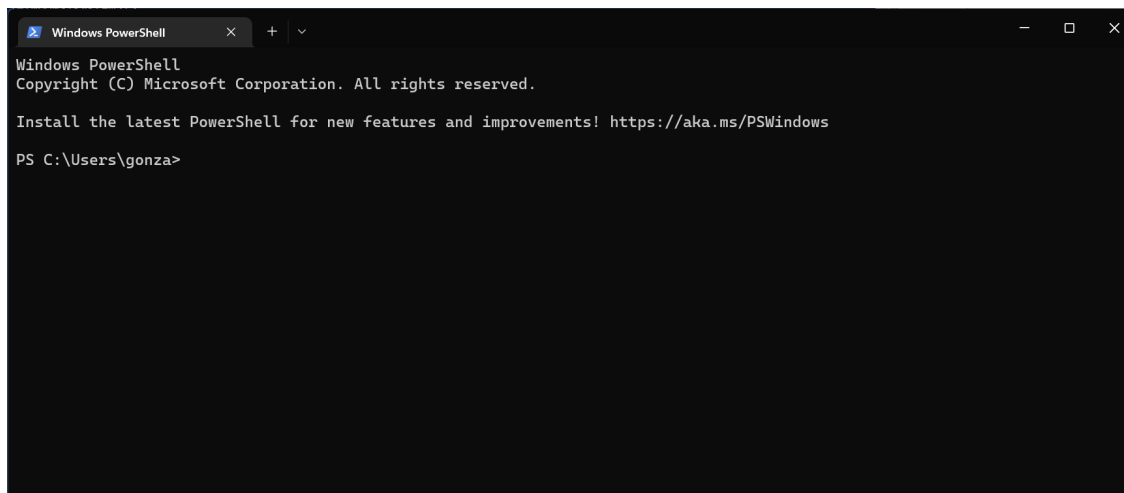
POWERSHELL

PowerShell

In Windows, to access PowerShell, you can enter PowerShell on the search bar to find the PowerShell app as shown in Screenshot 3. If you have a 64-bit device, when PowerShell is installed, PowerShell (x86) is also installed by default to allow you to run a 32-bit version of PowerShell (Microsoft, n.d.). Once PowerShell is found, you can click on the version of the app needed to open PowerShell (Screenshot 3 and Screenshot 4).



Screenshot 3: Searching for PowerShell



Screenshot 4: PowerShell

PowerShell Commands and Examples

As does CMD, PowerShell has a number of commands, each with its own set of arguments and options, that can be utilized. For this paper we only list the ones used in the scripts found in the on GitHub (<https://github.com/rwatson724/CMD-PowerShell-Scripts>) and only a high level and a brief description of how they work are provided. For a full list of PowerShell commands, options and extensive details visit SS64.com (Sheppard, An A-Z Index of Windows PowerShell commands, n.d.).

NEW-VARIABLE

On our journey to understanding PowerShell, we start with the creation of a new variable. All variables start with a **\$**.

NEW-VARIABLE (NV)	Syntax	Description
Create a new variable	New-Variable [-Name] <i>string</i> [[-value] <i>object</i>] [-scope <i>string</i>] [-description <i>string</i>] [-Visibility {Public Private}] [-option {None ReadOnly Constant Private AllScope}] [-force] [-passThru] [-whatif] [-confirm]	-Name The variable(s) name. -Value <i>object</i> The value assigned to the variable. -Scope <i>string</i> The scope where the variable is valid. Valid values are "Global", "Local", "Private" or "Script", or a number relative to the current scope. "Local" is the default. -Description <i>string</i> A description of the variable. -Option <i>string</i> Where the new variable should be visible/changeable: Valid values are: None Set no options. ("None" is the default.) ReadOnly The value of the variable cannot be changed except by using the Force parameter. However, the variable can be deleted. Constant The variable properties cannot be changed nor can the variable be deleted. Private Only available within the specified scope. AllScope The variable is copied to any new scopes that are created. -Force Override restrictions as long as security is not compromised. Allow the creation of a new variable that has the same name as an existing read-only variable. -Visibility {Public Private} Whether the variable is visible outside of the session in which it was created.

Sheppard op. cit

There are several ways to create a new variable. The *object* can use a cmdlet (function), a text string, a mathematical operation. In the example, below the new variable is created using a cmdlet. We created three date variables. Notice the **\$** that precedes the name of the variable.

```
$today_hyp = get-date -format "yyyy-MM-dd"
$today_noh = get-date -format "yyyyMMdd"
$today_uns = get-date -format "yyyy_MM_dd"
```

GET-DATE

GET-DATE retrieves the current date and time. With the various options available, we can decide if we want to display only the date or only the time. In addition, we can decide how to display the date and/or time.

GET-DATE	Syntax	Description
Retrieves the current date and time	Get-Date [[-date] <i>DateTime</i>] [-displayHint {Date Time DateTime}] {[-format <i>string</i>] [-uFormat <i>string</i>]} [-year <i>int</i>] [-month <i>int</i>] [-day <i>int</i>] [-hour <i>int</i>] [-minute <i>int</i>] [-second <i>int</i>] [CommonParameters]	-date <i>DateTime</i> By default, Get-Date returns the current system date and time. The -date parameter allows you to specify a specific date and time. -displayHint <i>DisplayHintType</i> Display only the Date, only the Time or the DateTime. -format <i>string</i> Display the date and time in the indicated format. -uFormat <i>string</i> Display the date and time in Unix format. -year -month -day -hour -minute -second Specify individual date/time components to be displayed.

Using the same example for NEW-VARIABLE, we see that we used the GET-DATE command to retrieve the current system date. In addition, we utilized the format option to create one date with hyphens, one date without hyphens and a date with underscores. Notice the use of capital 'MM' to indicate the two-digit numeric month. The format for month and minute is case sensitive. A lower case 'mm' is used for the two-digit minute (Sheppard, How-to: Standard DateTime Format patterns for PowerShell; n.d.).

```
$today_hyp = get-date -format "yyyy-MM-dd"
$today_noh = get-date -format "yyyyMMdd"
$today_uns = get-date -format "yyyy_MM_dd"
```

Note that if a format is not specified then GET-DATE returns the current date and time in the long date and long time format (Display 4).

```
PS C:\Users\gonza> get-date

Wednesday, February 8, 2023 1:48:49 AM
```

Display 4: Default GET-DATE Output

GET-VARIABLE and WRITE-HOST

GET-VARIABLE is used to retrieve the value of a variable that has been created.

GET-VARIABLE (GV)	Syntax	Description
Retrieve the variables	Get-Variable [-Name] <i>string</i> [] [-Include <i>string</i>] [-exclude <i>string</i>] [-valueOnly] [-scope <i>string</i>] [<i>CommonParameters</i>]	-Name The name of the variable(s). -include <i>string</i> Retrieve only the specified items. Wildcards are permitted. -exclude <i>string</i> Omit the specified items. Wildcards are permitted. -valueOnly Get only the value of the variable. -scope <i>string</i> The scope where the alias is valid. Valid values are "Global", "Local", "Private" or "Script", or a number relative to the current scope. "Local" is the default.

To retrieve the value of a variable created GET-VARIABLE or GV can be used to write. When retrieving the value of a variable the use of the \$ is not needed. Both commands produce the same result as seen in Display 5.

```
PS C:\Users\gonza> $today_hyp = get-date -format "yyyy-MM-dd"
PS C:\Users\gonza> get-variable today_hyp

Name                Value
----                -
today_hyp           2023-02-08

PS C:\Users\gonza> gv today_hyp

Name                Value
----                -
today_hyp           2023-02-08
```

Display 5: GET-VARIABLE and GV Default Output

By default, GET-VARIABLE displays both the name of the variable and the value. However, the use of the option -VALUEONLY will suppress displaying the variable name. Notice that the option is not case sensitive, nor does it have to appear after the variable (Display 6).

```
PS C:\Users\gonza> gv today_hyp -valueonly
2023-02-08
PS C:\Users\gonza> gv -VALUEONLY today_hyp
2023-02-08
```

Display 6: GV with -VALUEONLY Option

Another way to display the value of a variable is to use WRITE-HOST.

WRITE-HOST	Syntax	Description
Writes a string for display. Typically written to the console but can be directed to any host.	Write-Host [[-object] <i>Object</i>] [-noNewLine] [-separator <i>Object</i>] [-foregroundColor <i>ConsoleColor</i>] [-backgroundColor <i>ConsoleColor</i>] [<i>CommonParameters</i>]	-object <i>Object</i> Object to display, typically a string. -noNewLine Do not end with a newline character. -separator String to output between objects displayed on the console.

WRITE-HOST by default displays only the value (i.e., the name is not displayed) as seen in Display 7. Another difference between GV and WRITE-HOST is that GV did not require the use of \$ when specifying the variable; however, WRITE-HOST does require the \$ to precede the variable name.

```
PS C:\Users\gonza> write-host $today_hyp
2023-02-08
```

Display 7: WRITE-HOST Output

GET-LOCATION

To retrieve the current location, we use GET-LOCATION or GL.

GET-LOCATION (GL, PWD)	Syntax	Description
Retrieve and display the current location	Get-Location [-psDrive <i>string</i> []] [-psProvider <i>string</i> []] [-UseTransaction] [<i>CommonParameters</i>] Get-Location [-stack] [-stackName <i>string</i> []] [-UseTransaction] [<i>CommonParameters</i>]	-psDrive Retrieve the current location of the specified PowerShell drive. -stack Display locations in the default path stack -stackName Display locations in the specified path stacks.

Notice the two different syntax used to retrieve the current location. We can use GET-LOCATION (or GL) or we can use (GL).path; both yield the same results (Display 8).

```
PS C:\Users\gonza> gl

Path
----
C:\Users\gonza

PS C:\Users\gonza> (gl).path
C:\Users\gonza
```

Display 8: GET-LOCATION Output

GET-CHILDDITEM

The GET-CHILDDITEM command retrieves all the files and subfolders in the indicated path. Depending on the options used, we can filter for specific files. GET-CHILDDITEM has several aliases that can also be used: GCI, DIR and LS.

GET-CHILDDITEM (GCI)	Syntax	Description
Retrieves the item or child items in a folder.	Get-ChildItem [[-path] <i>string</i> []] [-literalPath] <i>string</i> []] [-Attributes <i>FileAttributes</i>] [[[-filter] <i>string</i>] [-include <i>string</i> []] [-exclude <i>string</i> []] [-FollowSymlink] [-Depth <i>UInt32</i>] [-Name] [-Directory] [-File] [-Hidden] [-ReadOnly] [-recurse] [-force] [-System] [-UseTransaction] [<i>CommonParameters</i>]	-path <i>string</i> The paths to the items from which content is to be retrieved. Wildcards are permitted. Default is the current directory (.). -literalPath <i>string</i> Like Path above, only the value is used exactly as typed (i.e., no wildcards). If the path includes any escape characters they are enclosed in single quotation marks. -Attributes {ReadOnly Hidden System Directory Archive Device Normal Temporary SparseFile ReparsePoint Compressed Offline NotContentIndexed Encrypted IntegrityStream NoScrubData} Use the following operators to combine attributes. ! NOT + AND , OR No spaces are permitted between an operator and its attribute. Abbreviations for commonly used attributes: D Directory H Hidden R Read-only S System -include <i>string</i> Include only the specified items from the path. e.g. 'May*'. Wildcards are permitted. This works with the -recurse parameter. -exclude <i>string</i> Omit the specified items from the path e.g. '*SS64*'. Wildcards are permitted. -filter <i>string</i> A string that is used to filter what is retrieved. Wildcards are permitted. -recurse Retrieve the items plus all child items of the location(s) (i.e., items in subdirectories).

For example, if we want to only list the zip file that has a specific naming convention within the current directory, then the option -PATH is used to point to the folder and the option -FILTER are used. In the sample below, -PATH indicates that the path stored in the variable **rcvd** is to be searched. In addition, the -FILTER option specifies that only files with the naming convention ***\$today_hyp\$.zip** are to be retrieved. Recall from a previous example that **today_hyp** is the current date in YYYY-MM-DD format; therefore, we are looking for zipped files that have the current date in the YYYY-MM-DD format.

```
get-childitem -path $rcvd -filter *$today_hyp*.zip
```

NEW-ITEM

NEW-ITEM allows us to create new items such as files and folders. When using NEW-ITEM (alias NI) to create a new item, we need to indicate the ITEMTYPE (alias IT). In order to create a new directory we use the option DIRECTORY.

NEW-ITEM (NI)	Syntax	Description
Create a new item (e.g., files, folders) in the namespace	New-Item [-name] <i>string</i> [-path <i>string</i>] [-force] [-credential <i>PSCredential</i>] [-itemType <i>string</i>] [-value <i>object</i>] [-whatif] [-confirm] [-UseTransaction] [<i>CommonParameters</i>]	-name <i>string</i> The name of the new item. -path <i>string</i> The path(s) to the items. Wildcards are permitted. A dot (.) is used to specify the current location. -itemType <i>string</i> Specified type of the new item: file, directory, SymbolicLink "it" is an alias for "itemType".

The following code creates a new directory (-IT DIRECTORY). If you do not specify the IT, the default is to create a file. Notice the three different ways to implement the logic to create a new folder name __TEMP that will reside under the

```
ni -path $PgmPath\__TEMP -it directory
ni -it directory __TEMP
ni -path $PgmPath -name __TEMP -it directory
```

REMOVE-ITEM

Comparable to the RMDIR and DEL commands in CMD, REMOVE-ITEM cmdlet in PowerShell allows us to remove specified items. The main difference is that RMDIR only removes directories and DEL only removes files while REMOVE-ITEM (aliases DEL, RI) allow us to remove directories or files.

REMOVE-ITEM (DEL, RI)	Syntax	Description
Removes the specified item	Remove-Item { [-path] <i>string</i>[] [-literalPath] <i>string</i>[] } [-include <i>string</i>[]] [-exclude <i>string</i>[]] [-filter <i>string</i>] [-stream <i>string</i>[]] [-recurse] [-force] [-whatif] [-confirm] [-credential <i>PSCredential</i>] [-UseTransaction] [<i>CommonParameters</i>] Remove-Item [-stream <i>string</i>] [<i>CommonParameters</i>]	-Path <i>string</i>[] The path (or paths) to the items to be removed. Wildcards are permitted. -LiteralPath <i>string</i> Like Path above, only the value is used exactly as typed (i.e., no wildcards). If the path includes any escape characters they are enclosed in single quotation marks. -include <i>string</i> Remove only the specified items from the path. e.g. "May*" Note: this only works when the path includes a wildcard character. -exclude <i>string</i> Do not remove the specified items from the Path e.g. "**SS64*" Note: this only works when the path includes a wildcard character. -recurse Delete the items plus all child items of the location (i.e., items in subdirectories). -force Override restrictions that prevent the command from succeeding.

Here we are deleting the temporary folder (__TEMP) and we indicate that we want to remove all the items in the folder along with any child items (i.e., subdirectories). In addition, we want to force the deletion of the __TEMP folder and all its childitems regardless of any restrictions placed on the folder.

```
remove-item $PgmPath\__TEMP -recurse -force
```

MOVE-ITEM

MOVE-ITEM cmdlet in PowerShell is akin to the MOVE command in CMD. They both move an item from one location to another.

MOVE-ITEM (MI)	Syntax	Description
Move an item from one location to another location.	<pre>Move-Item { [-path] string[] [-literalPath] string[] } [[-destination] string] [-include string[]] [-exclude string[]] [-filter string] [-force] [-credential PSCredential] [-PassThru] [-Confirm] [-WhatIf] [-UseTransaction] [CommonParameters]</pre>	-Path string[] The path (or paths) to the items to be moved. Wildcards are permitted. -LiteralPath string Like Path above, only the value is used exactly as typed (i.e., no wildcards). If the path includes any escape characters they are enclosed in single quotation marks. -destination The path to the location where the items are to be moved. Wildcards are permitted. The default is the current directory. Alias for "destination" is "dest". -include string Include only the specified items from the path. e.g. "May" Note: this only works when the path includes a wildcard character. -exclude string Exclude the specified items from the Path e.g. "SS64" Note: this only works when the path includes a wildcard character.

Here we illustrate the moving of the SAS data sets found in the __TEMP folder found at the location indicated by the **pgmpath** variable to a location specified by the **uzdir** variable. Notice that we use GCI to retrieve only the SAS data sets from the specified location and that we are also looking at subdirectories within the location.

```
gci $Pgmpath\__TEMP -recurse -include *.sas7bdat | move-item -dest $uzdir
```

WRITE-OUTPUT

WRITE-OUTPUT is equivalent to ECHO. This allows us to write an object to the console or with the use of additional commands we could write to an external file.

WRITE-OUTPUT (ECHO, WRITE)	Syntax	Description
Writes an object to the pipeline. If it is the last command in the pipeline, it writes it to the console.	<pre>Write-Output [-inputObject] object[] [CommonParameters]</pre>	-inputObject object The object(s) that is written to the pipeline. This could be a variable, command or an expression.

For the example below the value of the variable **uzdir** along with the additional text string "folder was created" is written to the console.

```
echo "- $uzdir folder was created"
```

```
PS C:\Users\gonza\Desktop\Conferences\Drafts\Commando\Unzip\Received> echo "- $uzdir folder was created"
- C:\Users\gonza\Desktop\Conferences\Drafts\Commando\Unzip\Received\2023-01-18 folder was created
```

Display 9: ECHO Writing to Console

OUT-FILE

ECHO by default writes to the console if it is the last command. However, if we want to redirect the output to a file, we can use OUT-FILE. If there are several items being redirected to the same file, but we don't want to overwrite the other information in the output file, we can use the -APPEND option to add to the existing output.

OUT-FILE	Syntax	Description
Send the output to a file.	Out-File [-filePath] <i>string</i> [[-encoding] <i>string</i>] [-append] [-width <i>int</i>] [-inputObject <i>psobject</i>] [-force] [-noClobber] [-whatif] [-confirm] [<i>CommonParameters</i>]	-filePath <i>path</i> The path to the output file. -append Add the output to the end of an existing file, instead of overwriting the file contents. -force Override restrictions that prevent the command from succeeding. -noClobber Does not overwrite (replace the contents) of an existing file. If both -append and -noClobber are specified, the output is appended. By default out-file, overwrites an existing file.

Continuing with the example from ECHO, we now want to redirect the message to an external file instead of writing it to the console. We can specify the OUT-FILE command along with the name of the file that the output should be written to. In addition, -APPEND is used so that any information already in the file is not overwritten.

```
echo "- $uzdir folder was created" | out-file $pgmpath\unzip_ps_$today_hyp.txt  
-append
```

TEST-PATH

Sometimes we need to check to see if a directory exists prior to proceeding to the next step. TEST-PATH allows us to check for that existence.

TEST-PATH	Syntax	Description
Checks to see if a path exists	Test-Path { [-path] <i>string</i> [] [-literalPath] <i>string</i> [] } [-pathType <i>TestPathType</i>] [-isValid] [-include <i>string</i> []] [-exclude <i>string</i> []] [-Filter <i>string</i>] [-credential <i>PSCredential</i>] [-UseTransaction] [<i>CommonParameters</i>]	-Path <i>string</i> [] The PowerShell path (or paths) tested to see if it exists. Wildcards are permitted. -LiteralPath <i>string</i> Like Path above, only the value is used exactly as typed (i.e., no wildcards). If the path includes any escape characters they are enclosed in single quotation marks. Note that currently Test-Path will return \$true if the path is a single space. To avoid this trim the string with trim().

In Display 10, we illustrate the use of TEST-PATH. A variable **currpath** is created to store the current location. In addition, a new variable **confpath** has been created to store the location of the folder we want to do some sort of process on. In the first iteration for the creation of **confpath**, the 'o' was left off and when TEST-PATH is executed, it fails to find the indicated path and returns a value of False. The second time we create **confpath**, we include the 'o' and execute TEST-PATH again and this time it finds the path and returns a value of True.

```

PS C:\Users\gonza\Desktop\Conferences\Drafts> $currpath = gl
PS C:\Users\gonza\Desktop\Conferences\Drafts> $confpath = "$currpath\Command"
PS C:\Users\gonza\Desktop\Conferences\Drafts> test-path $confpath.trim()
False
PS C:\Users\gonza\Desktop\Conferences\Drafts> $confpath = "$currpath\Commando"
PS C:\Users\gonza\Desktop\Conferences\Drafts> test-path $confpath.trim()
True

```

Display 10: TEST-PATH Output

IF/IF-ELSEIF-ELSE

Like CMD, conditional logic can help us determine which portions of the script need to be executed. If a specific condition is met, then the command(s) indicated are executed. Within PowerShell, the condition is placed in parentheses (), while the commands that are to be executed are placed within curly braces { }.

IF / IF-ELSEIF-ELSE	Syntax	Description
If the condition is met, then it executes the indicated command. If using IF-ELSEIF, then if condition is not met, then the ELSE command is executed.	<pre> if (condition) {commands_to_execute} [elseif (condition2) {commands_to_execute}] [else {commands_to_execute}] </pre>	-condition An expression that is evaluated to true or false. -commands_to_execute A PowerShell or an external command that is executed if the expression is true. Note that the parentheses () are placed around the condition that is to be checked, while the curly braces { } are placed around the commands that are to be executed.

Continuing with the example used for TEST-PATH, we can incorporate the condition into an IF statement, to confirm the path does exist before continuing to the next portion of the script. In the snippet of code below, we check for the existence of the path before we precede to the commands that are to be executed. Notice that commands to be executed can be additional IF/IF-ELSEIF-ELSE logic. Each command is enclosed within parentheses and each set of commands that correspond to IF or ELSEIF or ELSE are contained within the curly braces.

```

if ( Test-Path $rcvd.trim() )
{
    ## COMMANDS TO BE EXECUTED
    if ( $rcvdfile )
    {
        ## COMMANDS TO BE EXECUTED
        if ( -Not (Test-Path $uzdir.trim() ) )
        {
            ## COMMANDS TO BE EXECUTED
        }
        else
        {
            ## COMMANDS TO BE EXECUTED
        }
    }
    else
    {
        ## COMMANDS TO BE EXECUTED
    }
}

```

SORT-OBJECT

SORT-OBJECT allows us to sort the object by the specified property value (e.g., creationtime, lastwritetime, length, fullname). To find a complete list of properties, specify `gci | gm -membertype *property` (Sheppard, How-to: Some basic PowerShell principles - Objects, Methods and Properties, n.d.).

SORT-OBJECT (SORT)	Syntax	Description
Sorts the object by property value	Sort-Object [[-property] <i>Object</i> []] [-inputObject <i>psobject</i>] [-culture <i>string</i>] [-caseSensitive] [-unique] [-descending] [<i>CommonParameters</i>]	-Property <i>Object</i> The objects are sorted based on the values of the indicated properties (e.g., creationtime). Wildcards are permitted. -CaseSensitive Sort UPPER and lower case letters separately. -Descending Sort in descending order. The descending parameter applies to all properties.

For the following example, we are sorting by the **creationtime**.

```
sort-object -property creationtime
```

SELECT-OBJECT

With SELECT-OBJECT we are able to retrieve various properties about the specified object(s).

SELECT-OBJECT (SELECT)	Syntax	Description
Select properties from the specified object or set of objects	Select-Object [[-property] <i>Object</i> []] [-excludeProperty <i>string</i> []] [-expandProperty <i>string</i>] [-first <i>int</i>] [-last <i>int</i>] [-Skip <i>int</i>] [-unique] [-inputObject <i>psobject</i>] [<i>CommonParameters</i>]	-Property <i>Object</i> [] The property or properties that are to be selected. -ExcludeProperty <i>string</i> Properties that are not to be selected. -ExpandProperty <i>string</i> Select and expand the specified property. -First <i>int</i> Select <i>int</i> number of objects from the beginning of an array of input objects. -Last <i>int</i> Select <i>int</i> number of objects from the end of an array of input objects.

In this example, we are going retrieve a child item from the file, **rcvdfile**, in the indicated location, **rcvd**. However, we only want certain properties associated with the object. In this case we want the **creationtime** of the indicated object and we want the **creationtime** stored in the variable **fdate** in YYYY-MM-DD format (Display 11).

```
$fdate = $(gci $rcvd\$rcvdfile | select -expandproperty creationtime | get-date -f "yyyy-MM-dd")
```

```
PS C:\Users\gonza\Desktop\Conferences\Drafts\Commando\Unzip\Received> $fdate = $(gci $rcvd\$rcvdfile | select -expandproperty creationtime | get-date -f "yyyy-MM-dd")
PS C:\Users\gonza\Desktop\Conferences\Drafts\Commando\Unzip\Received> gv fdate -valueonly
2023-01-18
```

Display 11: Illustration of SELECT-OBJECT

FOREACH

Similar to CMD FORFILES command, FOREACH executes a set of commands for each object within a collection of objects. As FOREACH loops through each object, the object is temporarily stored in a variable (i.e., *item* in *collection*).

FOREACH	Syntax	Description
Loops through a set of input objects and performs an operation on each object.	ForEach [-Parallel] (<i>item</i> In <i>collection</i>) { <i>ScriptBlock</i> }	<i>-item</i> A variable that holds the current item in the loop <i>-collection</i> A collection of objects, such as filenames, registry keys or servernames <i>-ScriptBlock</i> A block of script or operation to run against each object.

In the following example, each object within the collection (i.e., **sasfiles**) is captured in the **file** variable and the set of commands within the scriptblock (i.e., the commands within the curly braces) are executed on each item (i.e., on each **file**)

```
ForEach ($file in $sasfiles) {  
    ## COMMANDS TO BE EXECUTED  
}
```

EXPAND-ARCHIVE

Prior to PowerShell 5.0, the ability to zip files was not part of the built-in set of cmdlets. Other alternatives needed to be used. (Sheppard, New-Zipfile, Expand-Zipfile, n.d.).

EXPAND-ARCHIVE	Syntax	Description
Extract files from a zipped (archived) file	Expand-Archive [-Path] <i>string</i> [-DestinationPath] <i>string</i> [-force] [-Confirm] [-WhatIf] [<i>CommonParameters</i>]	<i>-DestinationPath string[]</i> The path where the extracted files are to be saved. <i>-Path string[]</i> The path where the zipped (archived) file is saved.

In the snippet of code, we indicate we want to expand (or unzip) the file specified in the **rcvdfile** variable at the location indicated by the **rcvd** variable. The extracted files from the zipped file are saved to the **__TEMP** folder found at the location indicated by the **pgmpath** variable.

```
Expand-Archive $rcvd\$rcvdfile $pgmpath\__TEMP
```

In order to determine if you can use EXPAND-ARCHIVE, \$PSVersionTable cmdlet can be used (Display 12).

```
PS C:\Users\gonza\Desktop\Conferences\Drafts\Commando\Unzip> $psversiontable
```

Name	Value
-----	-----
PSVersion	5.1.22621.963
PSEdition	Desktop
PSCompatibleVersions	{1.0, 2.0, 3.0, 4.0...}
BuildVersion	10.0.22621.963
CLRVersion	4.0.30319.42000
WSManStackVersion	3.0
PSRemotingProtocolVersion	2.3
SerializationVersion	1.1.0.1

Display 12: Checking Version of PowerShell

Extra Tidbit: Quoting in PowerShell

Within PowerShell, you can use double or single quotes. Depending on how they are used could yield different outcomes. When setting a variable, the use of single quotes treats the string as verbatim text, and the value will be stored as is as seen with the return value of '\$pgmpath\received' in Display 13.

```
PS C:\Users\gonza\Desktop\Conferences\Drafts\Commando\Unzip> write-host $pgmpath
C:\Users\gonza\Desktop\Conferences\Drafts\Commando\Unzip
PS C:\Users\gonza\Desktop\Conferences\Drafts\Commando\Unzip> $rcvd = '$pgmpath\received'
PS C:\Users\gonza\Desktop\Conferences\Drafts\Commando\Unzip> gv rcvd -valueonly
$pgmpath\received
```

Display 13: Using Single Quotes when Creating a New Variable

However, if the purpose is to resolve the variable `pgmpath` when creating the new variable `rcvd`, then double quotes are needed as demonstrated in Display 14.

```
PS C:\Users\gonza\Desktop\Conferences\Drafts\Commando\Unzip> $rcvd = "$pgmpath\received"
PS C:\Users\gonza\Desktop\Conferences\Drafts\Commando\Unzip> gv rcvd -valueonly
C:\Users\gonza\Desktop\Conferences\Drafts\Commando\Unzip\received
```

Display 14: Using Double Quotes when Creating a New Variable

So, you may be asking if we need to use quotes when creating a new variable, how do we add quotes around the new variable. If you need to add quotations around a variable there are several ways to do this depending on whether you want single or double quotes. To add single quotes around the value, we need to enclose the single-quoted string within double quotes (Display 15). Note that in order for the variable `rcvdfile` to resolve the order in which we use single and double quotes is important. Notice in Display 16 that we enclosed the double-quoted string within single quotes, and this treated the value as verbatim text.

```
PS C:\Users\gonza\Desktop\Conferences\Drafts\Commando\Unzip> $_rcvdfile = "'$_rcvdfile'"
PS C:\Users\gonza\Desktop\Conferences\Drafts\Commando\Unzip> gv _rcvdfile -valueonly
'ddd2ed_code 20221028.zip'
```

Display 15: Adding Single Quotes Around a Variable Value – Correct Way

```
PS C:\Users\gonza\Desktop\Conferences\Drafts\Commando\Unzip> $_rcvdfile = '"$_rcvdfile"'
PS C:\Users\gonza\Desktop\Conferences\Drafts\Commando\Unzip> gv _rcvdfile -valueonly
"$rcvdfile"
```

Display 16: Adding Single Quotes Around a Variable Value – Incorrect Way

There are two ways to add double quotes around the value in the variable, the first way is to double the double quotes around the double-quoted string. The second way is to use the PowerShell escape character (```) to essentially mask the double quote. Refer to Display 17.

```
PS C:\Users\gonza\Desktop\Conferences\Drafts\Commando\Unzip> $_rcvdfile = "\"$_rcvdfile\""
PS C:\Users\gonza\Desktop\Conferences\Drafts\Commando\Unzip> gv _rcvdfile -valueonly
"ddd2ed_code 20221028.zip"
PS C:\Users\gonza\Desktop\Conferences\Drafts\Commando\Unzip> $_rcvdfile = "`"$_rcvdfile`"
PS C:\Users\gonza\Desktop\Conferences\Drafts\Commando\Unzip> gv _rcvdfile -valueonly
"ddd2ed_code 20221028.zip"
```

Display 17: Adding Double Quotes Around a Variable Value

For more details on quoting, visit Microsoft® documentation on PowerShell Scripting [about Quoting Rules](#) (Microsoft, n.d.).

SAMPLE USES

Now that we have gone over the basics for CMD and PowerShell, we can go through some use cases.

UNZIP FOLDERS

It is not uncommon to receive data via ZIP files. In order to use the data, we need to unpack the file. This is easy enough to accomplish manually by right clicking on the file in Windows Explorer, selecting Extract All from the right-click menu, and then indicating where to save the extracted files. However, you may want to be selective about your data extraction. For example, you may have particular subfolders and file types that you want to extract, leaving the remainder of the data in the zip untouched. Manual extraction via point and click can be tedious and is prone to human error (e.g., a data file could be overlooked). Harnessing the power of either CMD or PowerShell can help ensure you extract the files you need and place them in the desired location. Here we illustrate some steps on how to create a BAT or a PS1 script to unpack a ZIP file. Note that the full code for both the UNZIP.BAT and UNZIP.PS1 can be found in Github. (<https://github.com/rwatson724/CMD-PowerShell-Scripts>)

CMD

Our goal is to find a ZIP file with the current date in the file name and extract the data. If there is no ZIP file with the current date in the file name, then we need to find the latest ZIP created and attempt to unpack that file. For the purpose of this script, the assumption is that the date in the ZIP file name is one of the following formats YYYY-MM-DD, YYYY_MM_DD or YYYYMMDD.

Using CMD to create a script to extract data from a ZIP file, we start with determining the current date and setting the necessary environment variables (i.e., **today**, **runday**, **rcvd**). This was illustrated in the [SET section for CMD Commands and Examples](#). Once we have set the environment variables, we can begin searching for the ZIP file.

When searching for the ZIP file with the current date, we search in the indicated folder, **rcvd**, for a ZIP file that has the current date, **__today**. If a file is found, then the environment variable, **rcvdfile**, is set to the file name and we then jump to the **FoundFile** location. Note because we want the file with the latest date, we use /O:-D option and /B only returns the name.

```
set __today=%today%
for /F "delims=" %%I in ('DIR "%rcvd%\*%__today%.zip" /B /O:-D') DO (
set rcvdfile=%%I
goto FoundFile
)
```

This process is repeated for each date pattern. If there is no ZIP file found with the current date, then the file with the latest creation date in the **rcvd** folder is retrieved and it is saved in **rcvdfile**. Once the latest file is found, instead of jumping to **FoundFile**, we need to jump to the **FMTDate** location because we need to extract the last modified date of the file and format that date to desired format for the folder where the extracted files are saved.

```
for /F "delims=" %%I in ('DIR "%rcvd%\*.zip" /A:-D /B /O:-D /T:C') DO (
set rcvdfile=%%I
goto FMTDate
)
```

In order to extract the date from the file name found, we use FORFILES to retrieve the date and write the date to a temporary external file (___TEMPTEXT.TXT). The external file is then read back in so that the date stored in the file can be retrieved and reformatted.

```
if defined __rcvdfilename (
forfiles /P "%rcvd%" /M %__rcvdfilename% /C "cmd /c echo @fdate" > ___TEMPTEXT.TXT
for /F "tokens=1" %%A in (___TEMPTEXT.TXT) DO set fdate=%%A
)
```

After a ZIP file is found and the date for the folder name is determined (i.e., current date or latest modified date), additional checks are done to see if there is already a folder with that date. If no dated folder with the indicated date exists, then the script proceeds to create the date folder and writes a message to an external file. If there is already a dated folder, then we jump to the location **FolderExists** and a message is written to an external file and then we are directed to the location Done to exit.

```
if exist %fldr% goto :FolderExists
mkdir ___TEMP
mkdir %today%
echo - %today% folder was created >> "%CD%\unzip_%runday%.txt"

REM: ADDITIONAL CMD STATEMENTS ARE HERE - NOT SHOWN

:FolderExists
echo - This folder %today% already exists within %CD% > "%CD%\unzip_%runday%.txt"
echo - Either confirm received file is already unzipped or delete the folder >>
"%CD%\unzip_%runday%.txt"
goto Done
```

If the folder does not exist and then after the folder is created, we move onto the step that extracts the necessary data files from the ZIP files. The following code utilizes VBScript to unpack the ZIP file since it does not depend on any additional ZIP software and it stores the contents in the ___TEMP folder that was previously created. Original VBScript is found on StackOverflow (Williamson, n.d.) and has been modified to meet our needs.

```
:UnZipFile <ExtractTo> <newzipfile>
set vbs="%temp%\_.vbs"
if exist %vbs% del /f /q %vbs%
( echo strExtractTo = WScript.Arguments.Item(0^)
  echo strNewZipFile = WScript.Arguments.Item(1^)
  echo WScript.echo strExtractTo, strNewZipFile
  echo set objFSO = CreateObject("Scripting.FileSystemObject")
  echo set objShell = CreateObject("Shell.Application")
  echo set FilesInZip = objShell.Namespace(strNewZipFile^).items
  echo objShell.Namespace(strExtractTo^).CopyHere(FilesInZip^)
  echo set objFSO = Nothing
  echo set objshell = Nothing
) > %vbs%

for %%A in ("%~2") do cscript //nologo //B "%vbs%" "%~1" "%%~A"
if exist "%vbs%" del /f /q "%vbs%"
```

After the ZIP file is unpacked, data files (e.g., SAS7BDAT files) are searched for in the ___TEMP folder and are moved to the dated folder. Once all the data files are moved, the ___TEMP folder is deleted.

```
for /R "%CD%\___TEMP\" %%A in (*.sas7bdat) do move "%%A" "%CD%\%today%"

rmdir /s /q "%CD%\___TEMP"
```

PowerShell

Now we show how to achieve the same goal using PowerShell.

Using GET-DATE we determine the current date and save it to a variable. This was illustrated in the [GET-DATE section for PowerShell Commands and Examples](#). In addition, to the date we need to determine the current location (note that the PS script should be saved in the location where you want to run it from) and the location of where the zip files are stored and save these to variables as well (**pgmpath** and **rcvd**, respectively). Once we have set the environment variables, we can begin searching for the ZIP file.

We search in the indicated folder, **rcvd**, for a ZIP file that has the current date. If a file is found, then the variable, **rcvdfile**, is set to the file name and the variable, **fdate**, is set to the current date with hyphens (e.g., **today_hyp**). Note that each format for the current date is checked and if **rcvdfile** is still missing after all formats are checked, then the file with the latest creation date in the **rcvd** folder is retrieved and it is saved in **rcvdfile** and the creation date for that file is saved in **fdate**.

```
$rcvdfile = $(get-childitem -path $rcvd -filter *$today_hyp*.zip)
# ADDITIONAL POWERSHELL COMMANDS
if ( $rcvdfile ) { $fdate = $today_hyp }
else { $rcvdfile = gci -path $rcvd -filter *.zip | sort-object -property
creationtime | select -last 1 }
# ADDITIONAL POWERSHELL COMMANDS
if ( -not($fdate) ) { $fdate = $(gci $rcvd\$rcvdfile | select -expandproperty
creationtime | get-date -f "yyyy-MM-dd") }
```

Once a file is found and a date for the folder is set, then we need to check to see if the folder already exists and if it does not, then we need to create it. The creation of a new folder is illustrated in [NEW-ITEM section for PowerShell Commands and Examples](#).

After we determine if we can unzip the file (i.e., if the folder previously existed, it does not unzip the file), we can use the EXPAND-ARCHIVE cmdlet to unpack the ZIP file (refer to [EXPAND-ARCHIVE section for PowerShell Commands and Examples](#) for details). Recall that we only want data files (e.g., SAS7BDAT files) so the EXPAND-ARCHIVE cmdlet extracts all the files in the ZIP file to a __TEMP folder. After the ZIP file is unpacked, then data files are moved from __TEMP to the dated folder. Once all the data files are moved, the __TEMP folder is deleted.

```
Expand-Archive $rcvd\$rcvdfile $PgmPath\__TEMP
gci $PgmPath\__TEMP -recurse -include *.sas7bdatt | move-item -dest $uzdir
remove-item $PgmPath\__TEMP -recurse -force
```

By default, running PowerShell scripts is disabled. In order to enable the capability to execute PowerShell scripts, we need to execute the following command in the console and select 'Y' to confirm we want to run. This allows us to run scripts saved on our device. (Hemedinger, 2011)

```
Set-ExecutionPolicy RemoteSigned
```

BATCH SUBMITS

Another common task is to run programs in batch so that they run on a clean SAS session and the logs and lst files are saved. This can be achieved via Windows Explorer by right clicking on the program and selecting 'Batch Submit'. Although this is a good approach, it can sometimes 'muddy' up your folders since the log and lst files are saved to the same folder. What do we do if we want the log and lst files saved in a different folder or what if we need to run a bunch of programs? We don't want to have to manually move all those log and lst files nor do we want to right click and select 'Batch Submit' for a bunch of programs. With CMD and PowerShell, we can come up with a more efficient solution.

Single Files

There are several ways to run a single program in batch. We illustrate two techniques using CMD and one using PowerShell.

CMD

To run a single program and save the log and lst files to the correct folder locations, we need to make sure those folders exist. For this particular BAT file, it assumes that the folders for the log and lst files already exist.

We can then use SET command in CMD to set an environment variable that points to the executable file for SAS and the CONFIG file.

```
set sas="C:\Program Files\SASHome\SASFoundation\9.4\sas.exe" -CONFIG "C:\Program Files\SASHome\SASFoundation\9.4\nls\en\sasv9.cfg"
```

We can then use the SET command to create environment variables that point to the folders where the logs and lsts files are to be saved.

```
set plog=-log "%CD%\Logs\"
set plst=-print "%CD%\Lsts\"
```

Once we have those variables created, then we can use the SET command to create a pop-up window that allows us to enter the name of the program (without the '.sas' extension). Notice that in Display 18 we are prompted to enter a program name. In the example the name of the program is 'CATQ Function.sas', we can enter the program name without the extension. This program name is saved in the environment variable.

```
set pgm=
set /p pgm="Name of Program to be Executed - must be a complete name: "
```



```
Name of Program to be Executed - must be a complete name: catq function
```

Display 18: Pop-up Window Prompting for Program Name

Now that we have that last piece of information, the BAT file will put it together to signal that a SAS session is to be opened, the program is to be executed and the log and lst files are to be saved into the indicated locations. The complete code can be found on GitHub (<https://github.com/rwatson724/CMD-PowerShell-Scripts>).

```
%sas% -sysin "%CD%\%pgm%.sas" %plog% %plst%
```

But what if you don't want to be prompted to type the program name, especially if you have programs with long or hard to remember names. We can create a drag-and-drop batch file that will allow us to drag the program and "drop it" onto the BAT file.

This drag-and-drop script is similar to the previous BAT file with some slight differences. With this revised script, we no longer need to make sure the folders where the log and lst files are to be saved exist, logic has been added to ensure that they do.

```

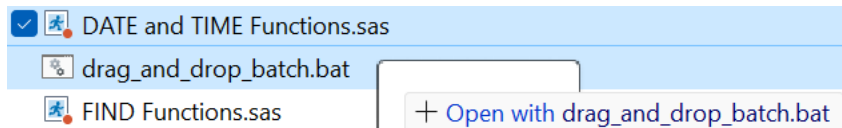
if exist "%~d1%~p1\Logs" (set LogFolder=%~d1%~p1\Logs) else (
    mkdir Logs
    set LogFolder=%~d1%~p1\Logs
)

```

In addition, instead of using a SET command to prompt for a program name, we use a FOR command to allow us to send in the name of the file.

```
for %%A in (*) do set pgm=%%~nA
```

The drag-and-drop BAT file allows us to select the program we want to execute and drag it on top of the BAT file (Display 19). Once the file is dropped onto the BAT file, a SAS session is opened, the program is executed and the log and lst files are saved to the appropriate location.



Display 19: Drag and Drop a SAS Program onto a BAT File

PowerShell

With PowerShell we are going to illustrate the drag-and-drop technique. However, this works a bit differently than the CMD technique. In fact, the drag-and-drop technique in PowerShell needs to work in tandem with CMD to achieve this. First, we describe the PowerShell script.

Within the PowerShell script, we need to define the necessary variables that will be used to execute the program starting with the name of the file. The **file** is read in as a parameter that is dropped onto a BAT file. Once the parameter is read into the PowerShell script the information, such as path (**pgmpath**) and filename (**pgmname**), associated with that file is extracted. In order to retrieve the filename without the extension, the BASENAME attribute is used.

```

$File = $args[0]
$PgmPath = (gi $File).path
$PgmName = (gi $File).basename

```

Once we have the necessary variables, we can check for the existence of the log and lst folders and if they do not exist, then they are created and a new variable is set that has the name of the log file in the appropriate folder (code is repeated for Lsts folder).

```

$logdir = "$PgmPath\Logs"
if ( -Not (Test-Path $logdir.trim()) )
{
    New-Item -Path $logdir -ItemType Directory
}
$LogPath = "$($logdir)\$PgmName.log"

```

Additional variables are set that point to the location of the SAS executable file and the SAS configuration file.

```

$sasexe = "C:\Program Files\SASHome\SASFoundation\9.4\sas.exe"
$sascfg = "C:\Program Files\SASHome\SASFoundation\9.4\nls\en\sasv9.cfg"

```

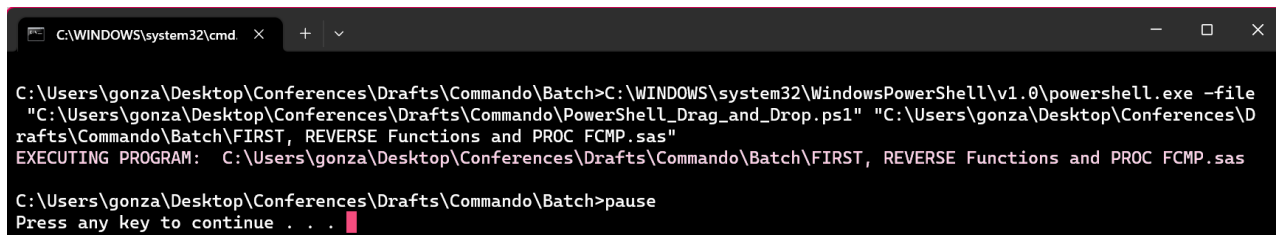
With all these variables, we can now build our command that will allow us to execute the SAS program that is dropped onto the BAT file.

```
Write-Host "EXECUTING PROGRAM: " $File
& "$sasexe" "$File" -NOLOGO -RSASUSER -CONFIG "$sascfg" -LOG "$logpath" -PRINT
"$lstpath" | Out-Null
```

Now that we have our PowerShell script written, we want to be able to execute a SAS program with the drag-and-drop technique. In order to achieve this, we need to utilize CLI. The script is pretty straightforward. We indicate where the executable file for PowerShell resides, then we indicate the full filename (including path and extension) of our PowerShell script. Note that this is preceded by the -FILE option. Lastly, we need to indicate that the PowerShell file is going to be passed a parameter and this is denoted with **%***.

```
%SystemRoot%\system32\WindowsPowerShell\v1.0\powershell.exe -file
"C:\Users\gonza\Desktop\Conferences\Drafts\Commando\PowerShell_Drag_and_Drop.ps1"
%*
```

Since this is a BAT file, the drag-and-drop technique works the same way it does as described in the CMD section above. The difference is that it is now going to execute a PowerShell script. For this particular script, a pause is included in the BAT file to show what is being executed by writing a message to the console (Display 20).



```
C:\WINDOWS\system32\cmd. x + v
C:\Users\gonza\Desktop\Conferences\Drafts\Commando\Batch>C:\WINDOWS\system32\WindowsPowerShell\v1.0\powershell.exe -file
"C:\Users\gonza\Desktop\Conferences\Drafts\Commando\PowerShell_Drag_and_Drop.ps1" "C:\Users\gonza\Desktop\Conferences\
rafts\Commando\Batch\FIRST, REVERSE Functions and PROC FCMP.sas"
EXECUTING PROGRAM: C:\Users\gonza\Desktop\Conferences\Drafts\Commando\Batch\FIRST, REVERSE Functions and PROC FCMP.sas
C:\Users\gonza\Desktop\Conferences\Drafts\Commando\Batch>pause
Press any key to continue . . .
```

Display 20: PowerShell Drag-and-Drop Technique

Right-Click Menu Item

But wait, there's more! What if we still want the right-click functionality but we need the log and lst files to go to the appropriate location and not be saved in the same folder as the program?

In order to do so, we need to update the registry. The following code should be saved as a '.reg' file (e.g., "Create-SAS Single Batch.reg"). (Schmidt, n.d.)

```
Windows Registry Editor Version 5.00

[HKEY_CLASSES_ROOT\*\shell\SAS Single Batch\command]
@="\"C:\\Users\\gonza\\Desktop\\Conferences\\Drafts\\Commando\\SAS Single Batch
DnD.bat\" \"%L\""
```

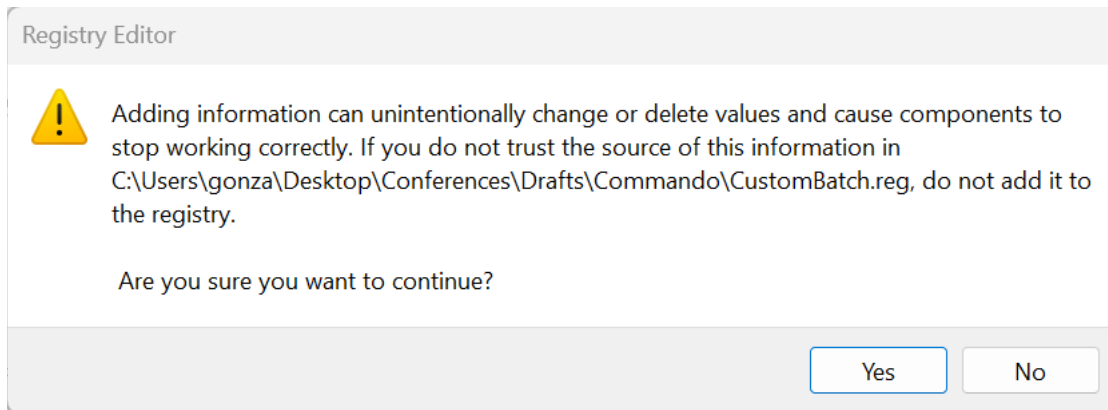
Note that "SAS Single Batch.bat" is the same script as "drag_and_drop_batch.bat"

To create the PowerShell drag-and-drop right-click menu item, we save the following as a '.reg' file (e.g., "Create-PowerShell DnD.reg"). (KyleMit, n.d.)

Windows Registry Editor Version 5.00

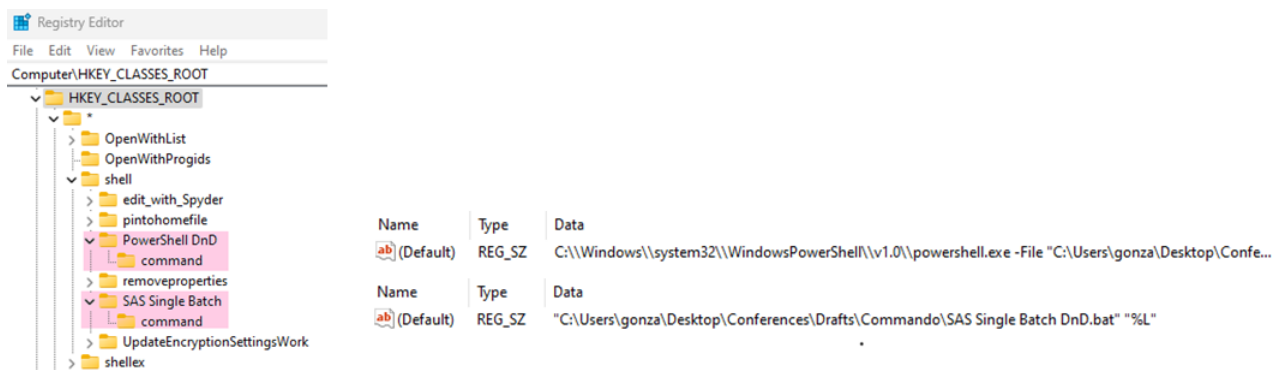
```
[HKEY_CLASSES_ROOT\*\shell\PowerShell DnD\command]
@="C:\\\\Windows\\\\system32\\\\WindowsPowerShell\\\\v1.0\\\\powershell.exe -
File
"C:\\Users\\gonza\\Desktop\\Conferences\\Drafts\\Commando\\PowerShell_Drag_and_
Drop.ps1\" \"%L\""
```

Once we have saved the reg files, we can double click on them to add them to the registry. Note you can only do this if you have the correct permissions and even then, you will get a message indicating that adding, changing or deleting can cause components to stop working correctly (Display 21).



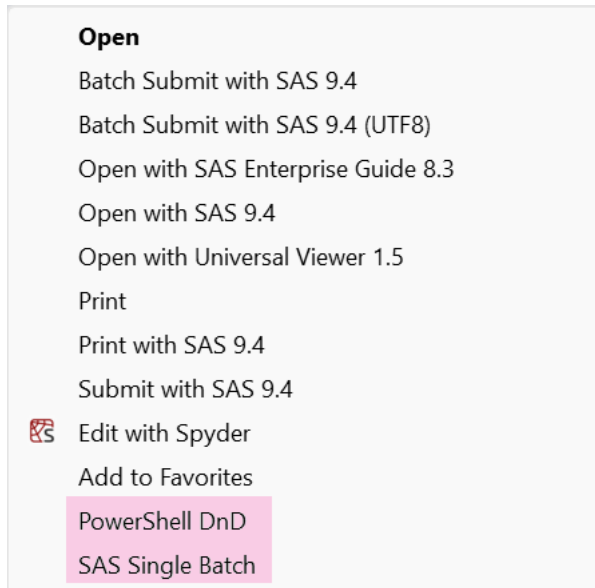
Display 21: Registry Editor Warning

Once the commands are added in the registry you will be able to see them in the Registry Editor (Display 22).



Display 22: Registry Editor with the Two New Right-Click Menu Items

Now when we right click on a SAS program, we have the options to either run via "PowerShell DnD" or "SAS Single Batch" (Display 23) and this way our logs and lsts files get written to the desired location versus writing to the location where the SAS program resides.



Display 23: New Right-Click Menu Items

Multiple Files

We have shown you how to execute a SAS program in batch using CMD and PowerShell and how to add a right-click menu option. While this is nice during the development phase of a program, it may not be ideal during production phase, when all the programs need to be executed. Now we turn our attention towards writing a script that will allow us to run all programs.

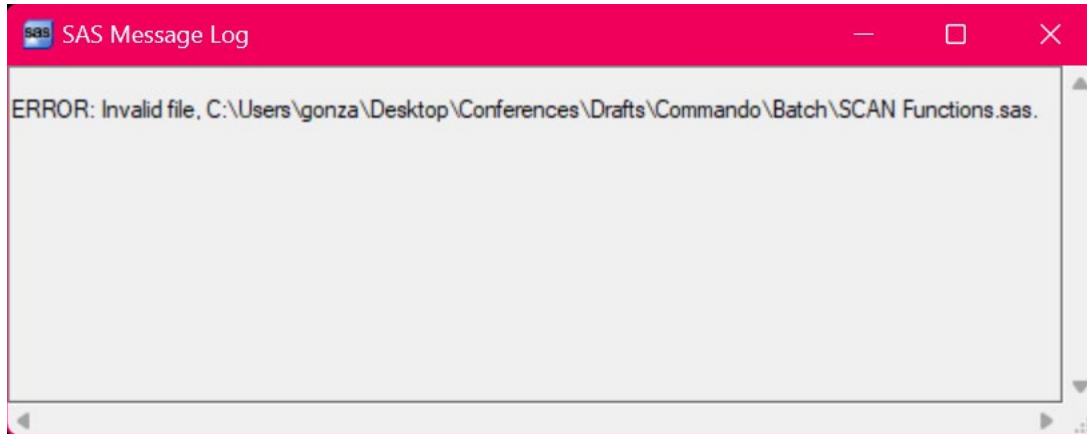
CMD – Manual Entry of Programs

Similar to the drag-and-drop technique, we check to see if the folders where the log and lst files are to be saved exists, and if they do not, we create them. We then set the environment variables, plog and plst. We also create the environment variables for the SAS executable file and SAS configuration file.

We then create a command to open SAS and execute the indicated program and write the log and lst file to the specified locations for every program that we want to run in batch.

```
%sas% -sysin "%CD%\ANYNOT Functions.sas" %plog% %plst%
%sas% -sysin "%CD%\CALL SORTC Routine.sas" %plog% %plst%
%sas% -sysin "%CD%\CATQ Function.sas" %plog% %plst%
%sas% -sysin "%CD%\COALESCE Functions.sas" %plog% %plst%
%sas% -sysin "%CD%\COMPRESS Functions.sas" %plog% %plst%
%sas% -sysin "%CD%\COUNT Functions.sas" %plog% %plst%
%sas% -sysin "%CD%\DATE and TIME Functions.sas" %plog% %plst%
%sas% -sysin "%CD%\FIND Functions.sas" %plog% %plst%
%sas% -sysin "%CD%\FIRST, REVERSE Functions and PROC FCMP.sas" %plog% %plst%
%sas% -sysin "%CD%\INDEX Functions.sas" %plog% %plst%
%sas% -sysin "%CD%\MISSING Routine and Functions.sas" %plog% %plst%
%sas% -sysin "%CD%\PATHNAME Function.sas" %plog% %plst%
%sas% -sysin "%CD%\PRX Functions.sas" %plog% %plst%
%sas% -sysin "%CD%\RESOLVE Function.sas" %plog% %plst%
%sas% -sysin "%CD%\SCAN Functions.sas" %plog% %plst%
%sas% -sysin "%CD%\TRANSLATE Functions.sas" %plog% %plst%
%sas% -sysin "%CD%\V Functions.sas" %plog% %plst%
```

The downside to this approach is that it is easy to mistype a program name and if a program is not found then you get an error message as shown in Display 24. In addition, you need to type all the program names in and only the ones included are executed.



Display 24: ERROR Message Produced When Program Cannot be Found

CMD – Automatic Entry of Programs

For the automatic approach of calling SAS programs to execute, everything is the same up to the point of where we specify the individual commands. Instead of the individual commands, we can use a FOR loop to loop through the current directory looking for SAS files and if they exist, then we execute the command to run the program for that file.

```
for %%I in (DIR "%CD%\*.sas" /B) DO (
  if exist %%I (%sas% -sysin "%%I" %plog% %plst%)
)
```

The advantage of this approach is that we don't have to worry about typing all the program names and making sure they are typed correctly. The disadvantage is that this will run **ALL** SAS programs in the current directory, so if there is an extraneous SAS program that is not to be executed, it either needs to be removed or additional logic needs to be added to the script so that it is skipped. Another disadvantage is that the programs may not be executed in the correct order.

PowerShell

The multi-batch version is similar to the single batch version in that it makes sure it has the appropriate folders for the log and list files and has the necessary variables for the SAS executable program and configuration file, but that is about it. Instead of having an argument passed via a CMD parameter to determine the SAS program name, we use a FOREACH loop to loop through the indicated path specified in **sasfiles** looking for files with '.sas' extensions. If the condition is met, then the commands are executed.

```
$sasfiles = $(Get-ChildItem -Path $PgmPath -Filter *.sas)
ForEach ($file in $sasfiles) {
  $saspgm = $File.basename
  $saspgmex = $File.Name
  $LogPath = "$($logdir)\$saspgm.log"
  $LstPath = "$($lstpath)\$saspgm.lst"
  Write-Host "EXECUTING PROGRAM: " $File
  $sasProgram = "$PgmPath\$saspgmex"
  & "$sasexe" "$sasProgram" -NOLOGO -RSASUSER -CONFIG "$sascfg" -LOG "$logpath"
  -PRINT "$lstpath" | Out-Null
}
```

UTILIZING THE COMMAND(O) LINE WITHIN SAS

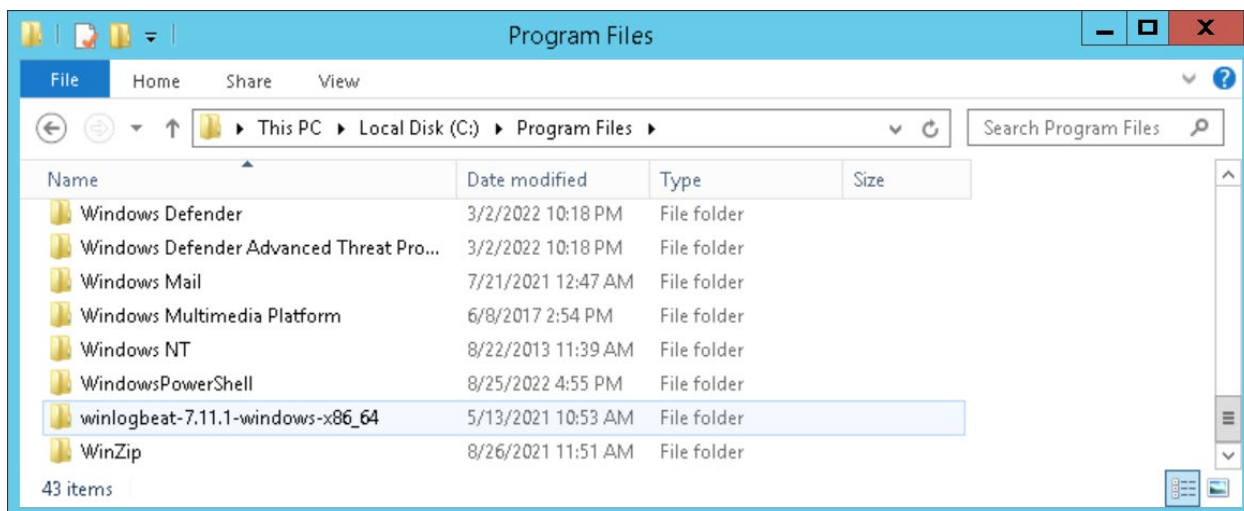
The fantastic material above describes how you can automate what would have been tedious manual steps in Windows with the clever use of CMD and PowerShell scripts. The ability to use command line scripts without accessing pay to play software is a valuable addition to our programming toolbox, and, non-programming staff can run these scripts. This section of the paper will provide three different similar solutions using CMD inside of SAS: creating an Excel and SAS directory listing using CMD in an unnamed pipe; writing and submitting a batch file with CMD and the X command; and zipping and unzipping within SAS using the X command and the WinZip CLI.

DIRECTORY LISTING VIA PIPE IN SAS

First, let's take a look at what the functionality we want looks like in PowerShell and CMD. In order to get a complete project listing we need details of all files in all folders and subfolders. This is the type of functionality that CLIs were built for. Both PowerShell and CMD (and other UNIX/LINUX/etc. shells) provide directory listings and a slew of subcommands, with similar, but not identical, syntax and functionality. It is advisable to consult documentation for each of these shells to find the correct syntax for the function you want to perform.

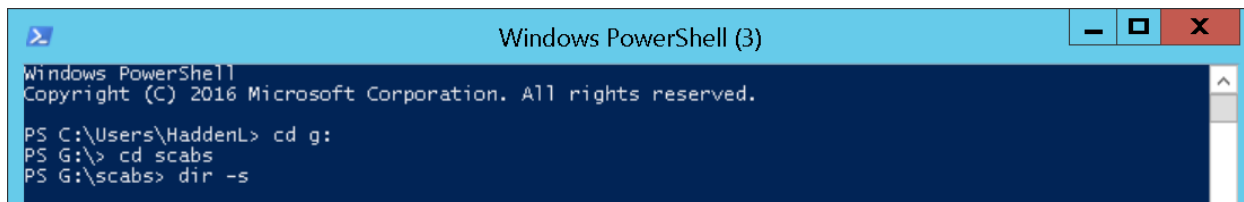
PowerShell Directory Listing

In Windows Server, PowerShell can be located in the Program Files folder under Windows PowerShell, as shown in Display 25. As noted in section [The Basics: PowerShell](#), you can search for PowerShell in the Windows Search menu item.



Display 25: Screenshot Showing the Location of Windows PowerShell

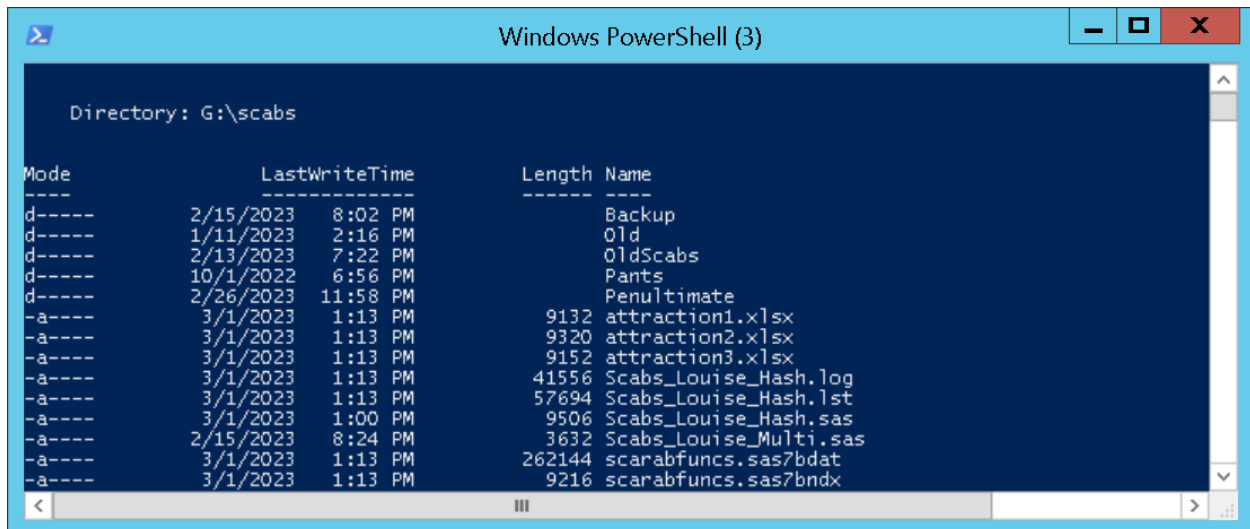
Windows PowerShell places the user in their Windows Users directory upon opening. You need to navigate to the drive and directory you want to obtain a listing for. This can be accomplished manually from the PowerShell prompt, or via the X or pipe commands in SAS. Some companies ban the use of PowerShell from other software packages: the second author's company is no exception. Manual use is permitted and demonstrated in Display 26, using a small directory with subdirectories.



Display 26: Navigating to a Directory and Obtaining a Full Directory Listing in PowerShell

The CD command is a short version of “change directory” – in PowerShell you can move from drive to drive and to a subdirectory in one step. It is shown as two steps here, moving from **c:\users\hadden** to the **g:** drive top level, then to a subdirectory **scabs**. Once we are in **scabs**, we issue a directory command (DIR) with a **-S** modifier (indicating we want the listing to include all subdirectories below the **scabs** directory).

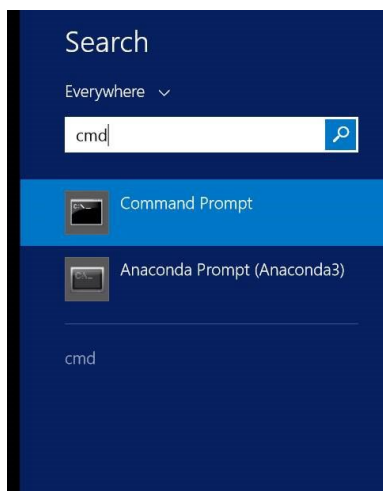
Below, we see the directory listing resulting from the commands. Since the second author’s company does not allow access to PowerShell directly from SAS, the author turns to the use of CMD instead to perform this within SAS.



Display 27: Screenshot of a Full Directory Listing in PowerShell

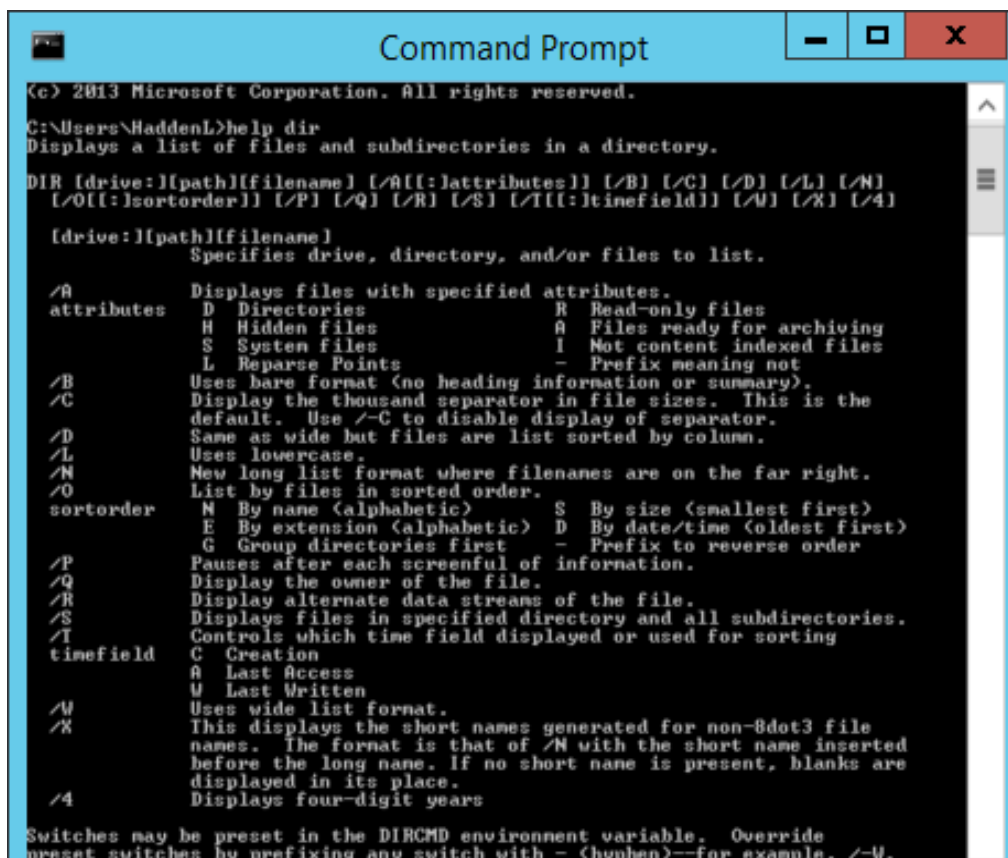
CMD Directory Listing

From Windows Server, we search for the system command shells available. Windows makes it somewhat difficult to locate the Command Prompt from anything other than the search command shown in Screenshot 1 and Screenshot 5. Once you find the Command Prompt (note that since Anaconda ALSO has a command shell available on our system, it pops up in the search) it is recommended that you pin the Command Prompt to the Start Menu and/or Task Bar for continued use.



Screenshot 5: Screenshot of Results of the Search Command in Windows Server

Once we have located the command prompt for CMD, we repeat the same process shown above for PowerShell, changing drives, navigating to the **scabs** subfolder, and creating a directory listing shown on the screen. Note that there are subtle but important differences between CMD and PowerShell. The first is that in CMD, you MUST change drives in a separate step, and the second is that instead of -S, /S is used. Please also note that case matters for some operating systems (Unix, Linux - /s may not be the same as /S. Your shell of choice will have a “HELP” feature (in Unix/Linux it is often “*man*”) – when in doubt, consult the documentation. For Windows platforms, typing HELP DIR will give you a quick digest of the syntax and subcommands for the DIR command.



```

(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\HaddenL>help dir
Displays a list of files and subdirectories in a directory.

DIR [drive:][path][filename] [/A[:attributes]] [/B] [/C] [/D] [/L] [/N]
  [/O[:sortorder]] [/P] [/Q] [/R] [/S] [/T[:timefield]] [/W] [/X] [/4]

  [drive:][path][filename]
    Specifies drive, directory, and/or files to list.

  /A      Displays files with specified attributes.
  attributes  D Directories                R Read-only files
               H Hidden files              A Files ready for archiving
               S System files              I Not content indexed files
               L Reparse Points            - Prefix meaning not

  /B      Uses bare format (no heading information or summary).
  /C      Display the thousand separator in file sizes. This is the
           default. Use /-C to disable display of separator.
  /D      Same as /W but files are list sorted by column.
  /L      Uses lowercase.
  /N      New long list format where filenames are on the far right.
  /O      List by files in sorted order.
  sortorder  N By name (alphabetic)          S By size (smallest first)
               E By extension (alphabetic)    D By date/time (oldest first)
               G Group directories first      - Prefix to reverse order

  /P      Pauses after each screenful of information.
  /Q      Display the owner of the file.
  /R      Display alternate data streams of the file.
  /S      Displays files in specified directory and all subdirectories.
  /T      Controls which time field displayed or used for sorting
  timefield  C Creation
               A Last Access
               W Last Written

  /W      Uses wide list format.
  /X      This displays the short names generated for non-8dot3 file
           names. The format is that of /N with the short name inserted
           before the long name. If no short name is present, blanks are
           displayed in its place.
  /4      Displays four-digit years

Switches may be preset in the DIRCMD environment variable. Override
preset switches by prefixing any switch with - (hyphen)--for example, /-W.

```

Display 28: Screenshot of Results of the Help Dir Command in Windows Server

Additionally, the results of a directory listing using PowerShell vs. CMD are different as seen in Display 27 and Display 29.

```

C:\>
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\HaddenL>g:
G:\>cd scabs
G:\Scabs>dir /s
Volume in drive G is Users New
Volume Serial Number is 68F2-F8FE

Directory of G:\Scabs

03/01/2023  02:16 PM    <DIR>          .
03/01/2023  02:16 PM    <DIR>          ..
03/01/2023  02:13 PM             9,132 attraction1.xlsx
03/01/2023  02:13 PM             9,320 attraction2.xlsx
03/01/2023  02:13 PM             9,152 attraction3.xlsx
02/15/2023  09:02 PM    <DIR>          Backup
01/11/2023  03:16 PM    <DIR>          Old
02/13/2023  08:22 PM    <DIR>          OldScabs
10/01/2022  06:56 PM    <DIR>          Pants
02/27/2023  12:58 AM    <DIR>          Penultimate
03/01/2023  02:13 PM             41,556 Scabs_Louise_Hash.log
03/01/2023  02:13 PM             57,694 Scabs_Louise_Hash.lst

```

Display 29: Screenshot of Results of the Search Command in Windows Server

The directory listing shown in Display 29 has the information we want, but it is on the screen, not in a file. CLIs provide the ability to redirect the results of a command to a text file. To redirect the results, we use the `>` symbol to create a text file from the `DIR /S` command. We confirm the creation of the text file by using the `TYPE` command. In Display 30, we show that the results of the `DIR /S` command are written to a file named “scabsdirlist.txt”. The text file can be processed manually to create a report, but there’s a better way.

```

G:\Scabs>dir /s > scabsdirlist.txt
G:\Scabs>type scabsdirlist.txt
Volume in drive G is Users New
Volume Serial Number is 68F2-F8FE

Directory of G:\Scabs

03/30/2023  12:57 AM    <DIR>          .
03/30/2023  12:57 AM    <DIR>          ..
03/01/2023  02:13 PM             9,132 attraction1.xlsx
03/01/2023  02:13 PM             9,320 attraction2.xlsx
03/01/2023  02:13 PM             9,152 attraction3.xlsx
02/15/2023  09:02 PM    <DIR>          Backup
01/11/2023  03:16 PM    <DIR>          Old
02/13/2023  08:22 PM    <DIR>          OldScabs
10/01/2022  06:56 PM    <DIR>          Pants
02/27/2023  12:58 AM    <DIR>          Penultimate
03/30/2023  12:57 AM             0 scabsdirlist.txt
03/01/2023  02:13 PM             41,556 Scabs_Louise_Hash.log
03/01/2023  02:13 PM             57,694 Scabs_Louise_Hash.lst
03/01/2023  02:00 PM             9,506 Scabs_Louise_Hash.sas
02/15/2023  09:24 PM             3,632 Scabs_Louise_Multi.sas
03/01/2023  02:13 PM             262,144 scarabfuncs.sas7bdat
03/01/2023  02:13 PM             9,216 scarabfuncs.sas7bndx
02/28/2023  01:58 AM             555 scarabsU1.csv
03/01/2023  01:24 PM             700 scarabsU2.csv
03/01/2023  02:03 PM             536 scarabsU3.csv
13 File(s)          413,143 bytes

```

Display 30: Screenshot of Redirecting the Results of the Search Command to a Text File in Windows Server

Creating SAS and Excel Output from a CMD Directory Listing in SAS

We have succeeded in creating a listing of files with CMD. Now the challenge is to replicate this process within SAS, in order to avoid typing, clicking, copying, and pasting. Additionally, we want to create both a SAS and Excel file from our directory listing, and create additional helpful variables in the process.

There are a number of ways to access system commands from within SAS. Two more commonly used methods are the X command and FILENAME and PIPE. This particular sample utilizes the FILENAME and PIPE method. All code samples provided below are snippets of larger programs, which are available in their entirety in the paper's GitHub page, <https://github.com/rwatson724/CMD-PowerShell-Scripts>. Previously, we demonstrated how to accomplish and redirect a directory listing with subdirectories in both Windows PowerShell and CMD. The FILENAME and PIPE commands allow us to redirect the results of a CLI command into a holding tank (pipe) that is accessible to SAS via a FILENAME statement.

```
libname dd '.';
filename yy '.\revisedcommando.txt';
filename dirlist pipe "dir ""&loc."" /s";
run;
```

The piped in text is edited in place to clean the input and prepare it for further manipulation in SAS with another FILENAME statement in program READDIRLIST2.sas (available in the paper's GitHub page, <https://github.com/rwatson724/CMD-PowerShell-Scripts>). As seen previously, the PowerShell and CMD outputs from a directory listing are not "fixed" record format; instead, records can present with multiple formats. The preprocessing categorizes the records into three primary types (main directory, sub directory, and filename) so they can be ingested appropriately and deletes empty and unnecessary records.

```
data temp1000 ;
  infile dirlist lrecl=1000 missover pad;
  input fool $char1000.;
  if fool='' then delete;
  fool=left(fool);
  if index(fool,'bytes')>0 then delete;

  seqnum=put(_n_,z8.);
  retain sumflag 0;
  dirflag=(substr(fool,1,9)='Directory');
  if substr(fool,25,5)='<DIR>' then rectype='sdirname';
  if substr(fool,40,1)='.' then delete;
  if substr(fool,38,1) ne ' ' then rectype='filename';
  if substr(fool,1,9)='Directory' and substr(fool,25,5) ne '<DIR>'
    then rectype='mdirname';
  sumflag=sumflag+dirflag;

run;

data temp2;
  file yy lrecl=300 pad;
  set temp1000;
  put rectype dirflag seqnum fool;
run;

data temp3;
  length cdate $ 10 time $ 6 ampm $ 2 csize $ 13 name directory $ 240
         part2 $ 20 filetype sizefmt $ 32;
  infile yy lrecl=300 missover pad firstobs=2;
  input rectype $ 1-8
  @;

  if rectype='filename' then input
    cdate $ 21-30
    time $ 33-37
    ampm $ 39-40
```

```

csize $ 45-58
name $ 60-300
;
if rectype='sdirname' then input
cdate $ 21-30
time $ 33-37
ampm $ 39-40
name $ 60-300
;
if rectype='mdirname' then input
name $ 34-275;

```

Variables created during processing include DIRECTORY (full path), FILENAME, SIZE, DATE, TIME, RECTYPEFILETYPE, and ORIGINAL_ORDER (to facilitate resorting of the data). The resulting file is exported to Excel (Screenshot 6) and saved as a permanent SAS data set (Screenshot 7).

	A	B	C	D	E	F	G	H	I	J	K
	original_order	directory	name	date	time	ampm	filetype	size	bytesize	sizefmt	rectype
2	2	G:\Scabs	attraction1.xlsx	3/1/2023	02:13	PM	Excel	9132 KB+	9KB	filename	
3	3	G:\Scabs	attraction2.xlsx	3/1/2023	02:13	PM	Excel	9320 KB+	9KB	filename	
4	4	G:\Scabs	attraction3.xlsx	3/1/2023	02:13	PM	Excel	9152 KB+	9KB	filename	
5	5	G:\Scabs	Backup	2/15/2023	09:02	PM	Directory			.	sdirname
6	6	G:\Scabs	Old	1/11/2023	03:16	PM	Directory			.	sdirname
7	7	G:\Scabs	OldScabs	2/13/2023	08:22	PM	Directory			.	sdirname
8	8	G:\Scabs	Pants	10/1/2022	06:56	PM	Directory			.	sdirname
9	9	G:\Scabs	Penultimate	2/27/2023	12:58	AM	Directory			.	sdirname
10	10	G:\Scabs	Scabs_Louise_Hash.log	3/1/2023	02:13	PM	Program log	41556 KB+	41KB	filename	
11	11	G:\Scabs	Scabs_Louise_Hash.lst	3/1/2023	02:13	PM	Program listing	57694 KB+	56KB	filename	
12	12	G:\Scabs	Scabs_Louise_Hash.sas	3/1/2023	02:00	PM	SAS program	9506 KB+	9KB	filename	
13	13	G:\Scabs	Scabs_Louise_Multi.sas	2/15/2023	09:24	PM	SAS program	3632 KB+	4KB	filename	
14	14	G:\Scabs	scarabfuncs.sas7bdat	3/1/2023	02:13	PM	SAS DS	262144 KB+	256KB	filename	
15	15	G:\Scabs	scarabfuncs.sas7bndx	3/1/2023	02:13	PM	UNKNOWN	9216 KB+	9KB	filename	
16	16	G:\Scabs	scarabsV1.csv	2/28/2023	01:58	AM	UNKNOWN	555 0-B+	1KB	filename	
17	17	G:\Scabs	scarabsV2.csv	3/1/2023	01:24	PM	UNKNOWN	700 0-B+	1KB	filename	
18	18	G:\Scabs	scarabsV3.csv	3/1/2023	02:03	PM	UNKNOWN	536 0-B+	1KB	filename	

Screenshot 6: Screenshot of Reading in the Results of CMD Directory Listing – Excel

	original_order	directory	name	date	time	ampm	filetype	size
1	2	G:\Scabs	attraction1.xlsx	03/01/2023	02:13	PM	Excel	9,132
2	3	G:\Scabs	attraction2.xlsx	03/01/2023	02:13	PM	Excel	9,320
3	4	G:\Scabs	attraction3.xlsx	03/01/2023	02:13	PM	Excel	9,152
4	5	G:\Scabs	Backup	02/15/2023	09:02	PM	Directory	
5	6	G:\Scabs	Old	01/11/2023	03:16	PM	Directory	
6	7	G:\Scabs	OldScabs	02/13/2023	08:22	PM	Directory	
7	8	G:\Scabs	Pants	10/01/2022	06:56	PM	Directory	
8	9	G:\Scabs	Penultimate	02/27/2023	12:58	AM	Directory	
9	10	G:\Scabs	Scabs_Louise_Hash.log	03/01/2023	02:13	PM	Program log	41,556
10	11	G:\Scabs	Scabs_Louise_Hash.lst	03/01/2023	02:13	PM	Program listing	57,694
11	12	G:\Scabs	Scabs_Louise_Hash.sas	03/01/2023	02:00	PM	SAS program	9,506
12	13	G:\Scabs	Scabs_Louise_Multi.sas	02/15/2023	09:24	PM	SAS program	3,632
13	14	G:\Scabs	scarabfuncs.sas7bdat	03/01/2023	02:13	PM	SAS DS	262,144
14	15	G:\Scabs	scarabfuncs.sas7bndx	03/01/2023	02:13	PM	UNKNOWN	9,216
15	16	G:\Scabs	scarabsV1.csv	02/28/2023	01:58	AM	UNKNOWN	556
16	17	G:\Scabs	scarabsV2.csv	03/01/2023	01:24	PM	UNKNOWN	700
17	18	G:\Scabs	scarabsV3.csv	03/01/2023	02:03	PM	UNKNOWN	536
18	19	G:\Scabs\Backup	myfuncs.sas7bdat	02/13/2023	08:14	PM	SAS DS	262,144
19	20	G:\Scabs\Backup	myfuncs.sas7bndx	02/13/2023	08:14	PM	UNKNOWN	9,216

Screenshot 7: Screenshot of Reading in the Results of CMD Directory Listing – SAS Data Set

The resulting SAS data set and Excel file can be manipulated for various purposes. The second author's company uses this program for data management and documentation.

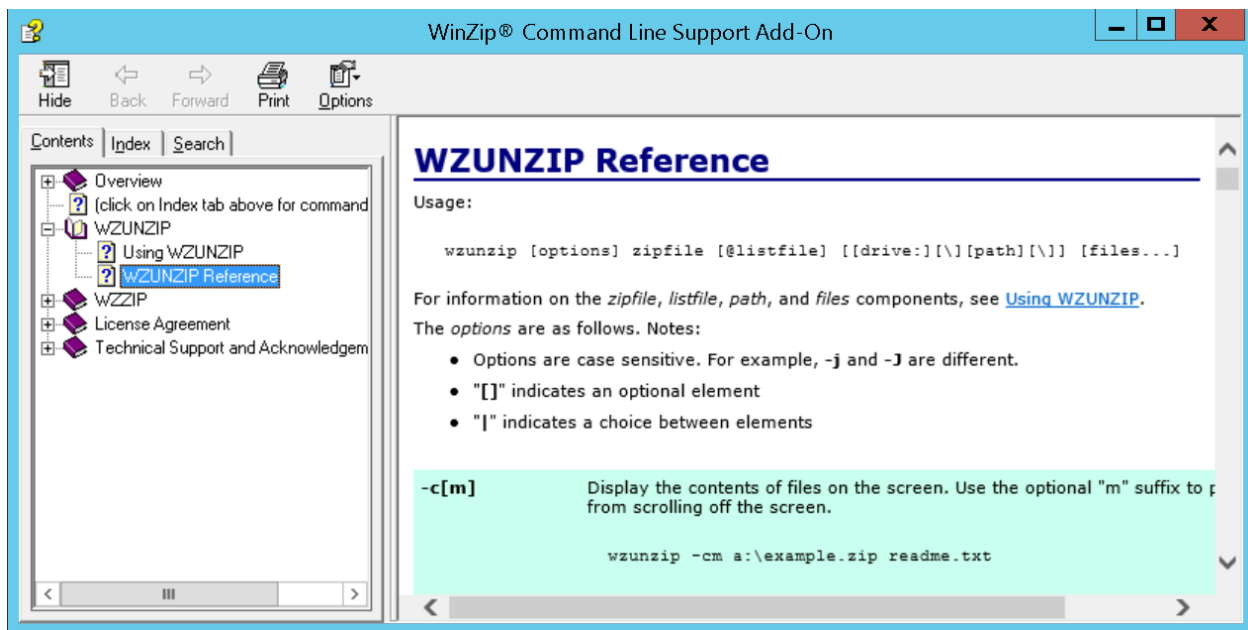
ZIPPING AND UNZIPPING VIA X COMMAND IN SAS

For this example, we chose to use the X command as opposed to a FILENAME and PIPE and batch file solution. Below is a simple sample of usage of the X command which copies all files with the XLSX extension from S:\DATA to G:\DATA.

```
X "copy s:\data\*.xlsx g:\data\*.xlsx";;
```

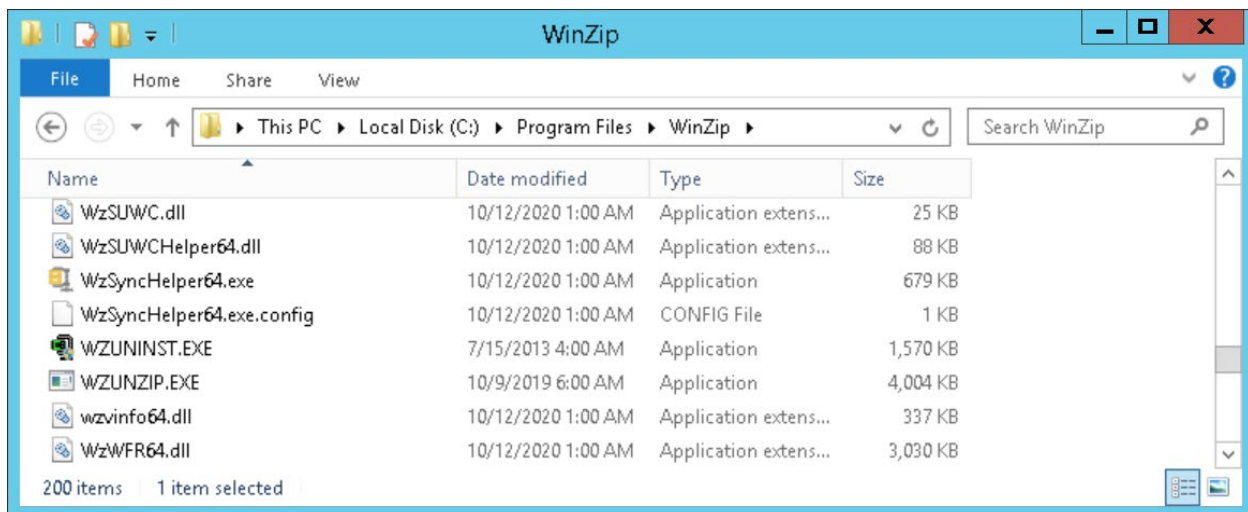
Zippping and unzipping within a SAS program is required for a large and complex project with monthly deliveries of multiple files, some of the files are very large, via Secure File Transfer Protocol (SFTP). Additionally, this process is used for regular archiving on the project. This example creates zipped deliverables in a specific domain on a Windows x64 cloud server, via the X command and the WinZip CLI. Other zip packages with CLIs such as 7zip can also be employed and have been successfully tested with this process (with appropriate syntax changes).

The first steps in operationalizing the zippping process were to consult the documentation for the WinZip CLI. Two basic commands in the WinZip CLI, WZZIP and WZUNZIP are used. You use WZZIP to create the zip file. Then, in order to obtain a listing of what is contained in the zip file you created, you need to temporarily unzip the file using WZUNZIP and redirecting the listing to an external file.



Screenshot 8: Screenshot of the WinZip CLI documentation

Next, just as we did for CMD and Windows PowerShell, we need to determine the location of the executables for WZZIP and WZUNZIP. If available, these executable files are typically located in a subfolder within in the c:\Program Files directory as seen in Display 31.



Display 31: Location of WinZip Executables

Using multiple X commands, we zip up our files, write a text file with a listing of the files in the zip file, read the text file in, and print the results for verification purposes. We are able to zip up exactly the files that we want with WinZip commands. Wildcards are allowed. Note that the "double double" quotes are required on our system as there are spaces in the pathnames. Note also the use of single period(".") and double periods("..") – this allows us to walk through a directory tree and navigate to the desired folders programmatically. This functionality can only be used when running in batch mode or through a batch program: interactive SAS platforms such as Enterprise Guide, SAS Studio, and the Display Manager require full pathnames and/or macro variables.

First, files are zipped using the X command to access the WinZip CLI's WZZIP executable. Note that all file pathnames are shortened for legibility.

```

* Zip up the files ;
x " "c:\program files\WinZip\wzzip"
  -a "\".. \Helpline_&fileyear.&filedate._allfiles.zip"
    "\".. \NHC_5star.xlsx"
    "\".. \Helpline_db_contents.rtf"
    "\".. \allfaclevel.sas7bdat"
    "\".. \allfaclevel_contents.rtf"
    "\".. \HICutpoints_State.rtf" " ";
. . .

```

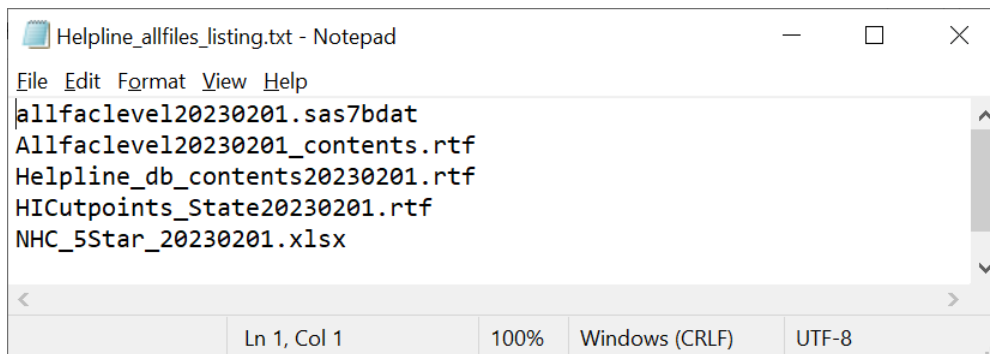
Second, a file listing is created by using the X command to access the WinZip CLI's WZUNZIP executable. The zip file just created is accessed again using -@ to create an external text file containing the contents of the zip file.

```

* write a text file with listing of files in zip;
x " "c:\program files\WinZip\wzunzip"
  -@"\".. \Helpline_allfiles_listing.txt"
    "\".. \Helpline_&fileyear.&filedate._allfiles.zip" " ";
run;
. . .

```

Lastly, the file listing is read into SAS and used to compare with the number and names of files meant to be included in the zip file to verify the zipping process.



Display 32: Verification of Zipping Process

CREATING AND RUNNING A BATCH SAS FILE VIA X COMMAND IN SAS

We saw previously how to operationalize a very slick batch process outside of SAS. Now, we are going to turn to creating a batch file and deploying it within a SAS file, via SAS data steps and the X command. For a very large and complex project, we are responsible for creating XML output files. Given the size of the input data, these 10 programs run for more than a day. To reduce the burden on a shared server, we need to run these programs from the command line, avoiding clogging up the Windows server. Note that as we saw previously in the CMD / PowerShell versions, we are able to add different “command line” parameters to the rows in the batch file – in this case we are setting the memory allocation to max. In order to make the program snippet legible the actual program path has been set to &loc. Note that the following SAS code creates the BAT file with the appropriate command with the necessary options to execute each program, and at the end it executes the BAT file via the X command. The contents of the BAT file can be viewed in Display 33.

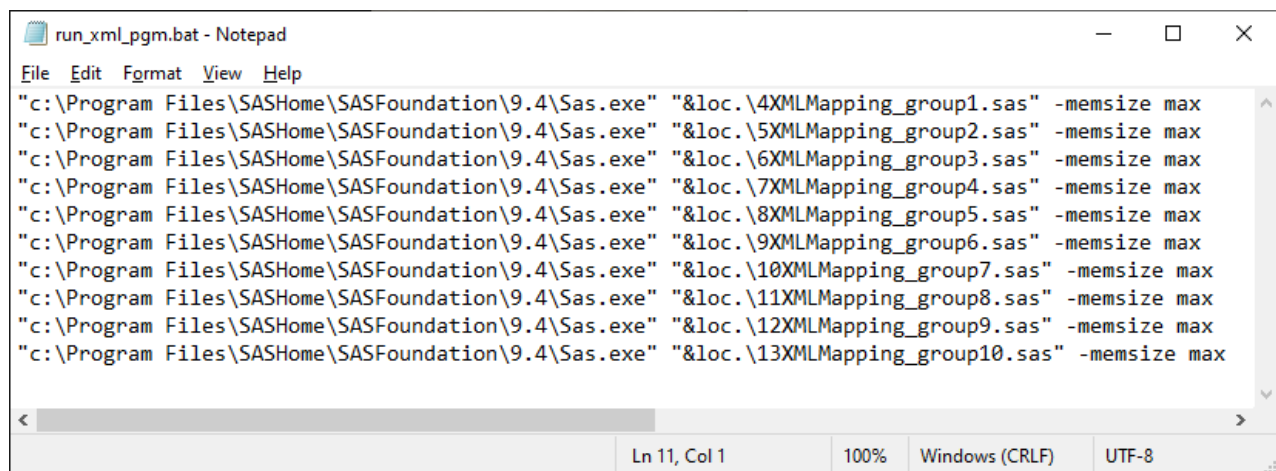
```

/*~ Make Batch File ~*/
filename bat 'run_xml_pgm.bat';
run;
data _null_;
  file bat;
  put ""c:\Program Files\SASHome\SASFoundation\9.4\Sas.exe""
    ""&loc.\4XMLMapping_group1.sas"" -memsize max";
  put ""c:\Program Files\SASHome\SASFoundation\9.4\Sas.exe""
    ""&loc.\5XMLMapping_group2.sas"" -memsize max";
  put ""c:\Program Files\SASHome\SASFoundation\9.4\Sas.exe""
    ""&loc.\6XMLMapping_group3.sas"" -memsize max";
  put ""c:\Program Files\SASHome\SASFoundation\9.4\Sas.exe""
    ""&loc.\7XMLMapping_group4.sas"" -memsize max";
  put ""c:\Program Files\SASHome\SASFoundation\9.4\Sas.exe""
    ""&loc.\8XMLMapping_group5.sas"" -memsize max";
  put ""c:\Program Files\SASHome\SASFoundation\9.4\Sas.exe""
    ""&loc.\9XMLMapping_group6.sas"" -memsize max";
  put ""c:\Program Files\SASHome\SASFoundation\9.4\Sas.exe""
    ""&loc.\10XMLMapping_group7.sas"" -memsize max";
  put ""c:\Program Files\SASHome\SASFoundation\9.4\Sas.exe""
    ""&loc.\11XMLMapping_group8.sas"" -memsize max";
  put ""c:\Program Files\SASHome\SASFoundation\9.4\Sas.exe""
    ""&loc.\12XMLMapping_group9.sas"" -memsize max";
  put ""c:\Program Files\SASHome\SASFoundation\9.4\Sas.exe""
    ""&loc.\13XMLMapping_group10.sas"" -memsize max";
run;

/*~ Run Batch File ~*/

x 'run_xml_pgm.bat';

```



Display 33: Batch File Written and Deployed Within SAS

CREATING AND RUNNING A COMMAND LINE BATCH FILE VIA DATA _NULL_ AND FILENAME PIPE IN SAS

We have demonstrated two methods of using SAS to utilize command line processing. There are additional methods, such as %SYSTASK, but exploring all possible methods is beyond the scope of this paper and presentation. In this example, we explore using a filename pipe within a DATA _NULL_ step to deploy batch commands in CMD. This routine has great potential for looping through a file programmatically, submitting

batch commands on the fly, without the need for “double double” quotes. The program snippet below creates a variable that contains a command or expression (the same one used in the directory listing example in this case), then assigns that command to a pipe device with the FILEVAR option. The command executes until EOF, leaving the option to have multiple records to execute multiple commands. The output of the command is echoed to the log with the put _INFILE_ statement and can be seen in Display 34, comingled with the lst via PROC PRINTTO. If it looks familiar, it is indeed the same output as yielded by the directory listing example.

```
proc printto log="batchpipetest.txt" print="batchpipetest.txt" new;
data _null_;
  f = 'dir "G:/SCABS"/s';
  infile ff PIPE FILEVAR=f end=EOF;
  do until(eof);
    input;
    put _infile_;
  end;
  stop;
run;

proc printto;
run;
```

batchpipetest.txt - Notepad

File Edit Format View Help

```
25      input;
26      put _infile_;
27      end;
28
29      stop;
30      run;
```

NOTE: The infile FF is:
 Unnamed Pipe Access Device,
 PROCESS=dir "G:/SCABS"/s,RECFM=V,LRECL=32767

Volume in drive G is Users New
 Volume Serial Number is 68F2-F8FE

Directory of G:\SCABS

04/03/2023	10:54 AM	<DIR>	.
04/03/2023	10:54 AM	<DIR>	..
04/02/2023	05:42 PM		9,133 attr1.xlsx
04/02/2023	04:59 PM		9,075 attr2.xlsx
04/02/2023	04:58 PM		9,045 attr3.xlsx
04/02/2023	02:45 PM	<DIR>	Backup
04/03/2023	10:54 AM		0 batchpipetest.log
04/03/2023	10:54 AM		863 batchpipetest.sas
04/03/2023	10:54 AM		0 batchpipetest.txt
04/02/2023	02:35 PM	<DIR>	Old
02/13/2023	08:22 PM	<DIR>	OldScabs
10/01/2022	06:56 PM	<DIR>	Pants
02/27/2023	12:58 AM	<DIR>	Penultimate

Ln 1, Col 1 100% Windows (CRLF) UTF-8

Display 34: Batch File Within SAS via FILENAME PIPE and Data _null_

CONCLUSION

The Windows Command Line Interface (CLI / CMD) and PowerShell operating systems can greatly enhance and improve utility of various operations, including custom scripts to zip and unzip files both with and without the use of an external zip application, create Excel and SAS directory listings, and create and submit batch SAS scripts. We hope you will explore the valuable tips we have demonstrated, and make the command line a part of your programming toolbox and put a little “power” into your code.

Full code for examples shown is available on GitHub: <https://github.com/rwatson724/CMD-PowerShell-Scripts>. Please feel free to contact us with any questions you may have or for further information.

Disclaimer: Not all companies allow their employees to access command line interfaces of any kind or the registry. You may need to work with your IT department to permit curated access to these tools or develop the script or right-click menu item. For example, the second author's company allows for the registration of named executables that allows the use of batch files.

REFERENCES

- Computer Hope. (n.d.). *Command line*. Retrieved Nov 2022, from Computer Hope: <https://www.computerhope.com/jargon/c/commandi.htm>
- Computer Hope. (n.d.). *Command-line interpreter*. Retrieved Nov 2022, from Computer Hope: <https://www.computerhope.com/jargon/c/comminte.htm>
- Hadden, L. (2012). Put a Little Zip in Your SAS® Program. Orlando, FL: SAS Global Forum. Retrieved from <https://support.sas.com/resources/papers/proceedings12/214-2012.pdf>
- Hadden, L. (2017). Get Smart! Eliminate Kaos and Stay in Control – Creating a Complex Directory Structure with the DLCREATEDIR Statement, SAS® Macro Language, and Control Tables. Long Beach, CA: WUSS. Retrieved from https://www.lexjansen.com/wuss/2017/43_Final_Paper_PDF.pdf
- Hemedinger, C. (2011, Sep 12). *Running Windows PowerShell Scripts*. Retrieved Nov 2022, from SAS Blogs: <https://blogs.sas.com/content/sasdummy/2011/09/12/running-windows-powershell-scripts/>
- How to Add New Options to Right Click Menu in Windows*. (n.d.). Retrieved Feb 2023, from wikiHow: <https://www.wikihow.com/Add-New-Options-to-Right-Click-Menu-in-Windows>
- KyleMit. (n.d.). *Execute Powershell Script on Right Click in Windows Explorer*. Retrieved Feb 2023, from GitHub Gist: <https://gist.github.com/KyleMit/978086ae267ff5be17811e99c9607986>
- Microsoft. (n.d.). *about_Quoting_Rules*. Retrieved Jan 2023, from Microsoft Technical Documentation: https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_quoting_rules?view=powershell-7.3
- Microsoft. (n.d.). *Command-line reference A-Z*. Retrieved Jan 2023, from Microsoft Technical Documentation: [https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-xp/bb490890\(v=technet.10\)](https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-xp/bb490890(v=technet.10))
- Microsoft. (n.d.). *Starting Windows PowerShell*. Retrieved Feb 2023, from Microsoft Technical Documentation: <https://learn.microsoft.com/en-us/powershell/scripting/windows-powershell/starting-windows-powershell?view=powershell-7.3>
- Microsoft. (n.d.). *What is PowerShell?* Retrieved Feb 2023, from Microsoft Technical Documentation: <https://learn.microsoft.com/en-us/powershell/scripting/overview?view=powershell-7.3>
- Schmidt, R. (n.d.). *Windows: How to add batch-script action to Right Click menu*. Retrieved Nov 2022, from StackExchange: <https://superuser.com/questions/444726/windows-how-to-add-batch-script-action-to-right-click-menu>
- Sheppard, S. (n.d.). *An A-Z Index of Windows CMD commands*. Retrieved Jan 2023, from SS64.com: <https://ss64.com/nt/>
- Sheppard, S. (n.d.). *An A-Z Index of Windows PowerShell commands*. Retrieved Feb 2023, from SS64.com: <https://ss64.com/ps/>
- Sheppard, S. (n.d.). *How-to: Pass Command Line arguments (Parameters) to a Windows batch file*. Retrieved Jan 2023, from SS64.com: <https://ss64.com/nt/syntax-args.html>
- Sheppard, S. (n.d.). *How-to: Some basic PowerShell principles - Objects, Methods and Properties*. Retrieved Feb 2023, from SS64.com: <https://ss64.com/ps/syntax-objects.html>
- Sheppard, S. (n.d.). *How-to: Standard DateTime Format patterns for PowerShell.* Retrieved Jan 2023, from SS64.com: <https://ss64.com/ps/syntax-dateformats.html>
- Sheppard, S. (n.d.). *How-to: Variables and Operators (add, subtract, divide...)*. Retrieved Feb 2023, from SS64.com: <https://ss64.com/ps/syntax-variables.html>
- Sheppard, S. (n.d.). *How-to: Windows Environment Variables*. Retrieved Jan 2023, from SS64.com: <https://ss64.com/nt/syntax-variables.html>
- Sheppard, S. (n.d.). *New-Zipfile, Expand-Zipfile*. Retrieved Feb 2023, from SS64.com: <https://ss64.com/ps/zip.html>

SS64. (2022). Retrieved Nov 2022, from SS64: <https://ss64.com/>

Williamson, M. (n.d.). *creating batch script to unzip a file without additional zip tools*. Retrieved Oct 2022, from Stack Overflow: <https://stackoverflow.com/questions/21704041/creating-batch-script-to-unzip-a-file-without-additional-zip-tools>

ACKNOWLEDGEMENTS

We owe a debt of gratitude to the helpful compatriots on the SAS-L listserv; in particular, Bart Jablonski and his encyclopedic knowledge of SAS.

RECOMMENDED READING

- *Base SAS® Procedures Guide*
- *SAS® For Dummies®*

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Richann Jean Watson
DataRich Consulting
richann.watson@datarichconsulting.com

Louise S. Hadden
Abt Associates, Inc.
louise_hadden@abtassoc.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Any brand and product names are trademarks of their respective companies.