Advanced Scripting   
Data presented as Plaintext, CSV

Last Updated: 1/20/2024 11:47 PM Version 1  
Document Prepared for: CYBER360 Student

# Name Click here to enter name ID Click here to enter id

# Instructions

Answer all questions directly in this document. You will save and upload this completed document as your homework submission.

# Overview

There are several popular ways to organize text-based data within files and streams, and there are PowerShell commands and modules to parse most of them.

# Requirements

* psfiles.zip

# Setup

psfiles.zip

# Task 1—Working with plaintext data

## Steps

1. Change your current working directory to your psfiles/data folder.
2. The colors.txt file contains a list of color names. There is one name per line. Read the file into a variable with the **Get-Content** cmdlet:   
   $colors=Get-Content colors.txt
3. Use **Measure-Object** to see how many colors there are:   
   $colors|measure
   1. How many colors are in the file? Click or tap here to enter text.
4. You can easily filter the colors by using the **Select-String** cmdlet. **Select-String** uses regular expression matching. It will return any string that matches an expression, but filter out strings that don’t match. Use **Select-String** to list all the colors that contain the word green.   
   $colors|Select-String green
   1. How many colors have the word green in them? Click or tap here to enter text.
5. Use the **-like** operator to perform basic pattern matching. To get all the colors that start with the letters ‘co’ enter:  
   $colors|? {$\_ -like 'co\*'}
   1. How many colors start with ‘co’? Click or tap here to enter text.
6. Get the first ten colors in the colors.txt file:  
   Get-Content .\colors.txt -Head 10
   1. What is the 10th color in the file? Click or tap here to enter text.
7. Get the last five colors in the file  
   Get-Content .\colors.txt -Tail 5
   1. What color is 5th from last in the file? Click or tap here to enter text.
8. List the contents of the colors file one page (one screenful) at a time:   
   Get-Content .\colors.txt|Out-Host -Paging
9. Create a new text file named **firstlast.txt** with the first ten and the last ten colors. Instead of using redirection operators, we’ll use some handy cmdlets to create and append to the file.
   1. First grab the first 10 items and store in the output file:   
      Get-Content .\colors.txt -Head 10 |Set-Content firstlast.txt
   2. Check to see if it worked  
      Get-Content .\firstlast.txt
   3. Next grab the last ten items and append it to the file.   
      Get-Content .\colors.txt -Tail 10 |Add-Content firstlast.txt
   4. Check to see if the colors were added:  
      Get-Content .\firstlast.txt
   5. Grab the 11th through the 20th colors from the file and store them in **firstlast.txt**:   
      (Get-Content .\colors.txt)[10..19]|set-content .\firstlast.txt
   6. Describe what is in firstlast.txt now: Click or tap here to enter text.
   7. Finally, clear the **firstlast.txt** file:   
      Clear-Content .\firstlast.txt
   8. Now what is in **firstlast.txt**? Click or tap here to enter text.

# Task 2—Working with “comma-separated-value” (CSV) data

## Steps

1. Examine the **gems.csv** file as “raw” text data:   
   **Get-Content .\gems.csv**
2. Now load and parse the spreadsheet data from **gems.csv** :   
   $g=Import-Csv .\gems.csv
3. View the imported data:   
   $g|ft
4. View the resulting data type:   
   $g[0].GetType().Name
   1. What is the data type of each item? Click or tap here to enter text.
5. Save a list of all the minerals that have a hardness of 5 or more into a smaller CSV file named **hardgems.csv** that only contains two spreadsheet columns: the mineral name, and its hardness in order of hardest to softest. On one line enter:   
   Import-Csv gems.csv |Where {+($\_.hardness) -ge 5}|sort hardness -desc |select Mineral,Hardness|Export-Csv hardgems.csv   
   (Think about this one. Exactly how is the data processed between each pipe in the pipeline? Try executing the pipeline one command at a time, and watch how each filter transforms the data:   
   Import-Csv gems.csv   
   Import-Csv gems.csv |Where {+($\_.hardness) -ge 5}   
   Import-Csv gems.csv |Where {+($\_.hardness) -ge 5}|sort hardness -desc   
   and so forth.)
6. View the contents  
   cat hardgems.csv
   1. Did it work? Click or tap here to enter text.
7. Diamond, with a hardness of 10 ended up at the bottom of the list. This is because the hardness was sorted “alphabetically” (“asciibetically?”) as a string, rather than sorted as a number! We need to fix that. Hit your “up-arrow” key to get that long command pipeline from your command history, then move the cursor back with “left-” or “right-arrow” keys and insert another command in the pipeline:   
   Import-Csv gems.csv |Where {+($\_.hardness) -ge 5}|% {$\_.hardness = +$\_.hardness; $\_} |sort hardness -desc |select Mineral,Hardness|Export-Csv hardgems.csv
   1. Is it sorted correctly this time? Click or tap here to enter text.
   2. In your own words, describe what the extra command in step 7 did to fix the problem. Click or tap here to enter text.

# Task 3—Working with tab-delimited CSV data

The “CSV” commands also work with files that are not comma separated. Let’s load and parse a file where the spreadsheet columns are separated with a tab.

## Steps

1. Import the tab separated file **RushSongs.txt**. The delimiter is a special control character, so we need to use its escape sequence when we specify the delimiter. Enter the following, but note that the *backtick* (or *grave accent*) **`** is not the same as a single quote. (On most keyboards, this backtick character is at or near the top-left corner, above [Tab] and left of the [1] key.):  
   $s=Import-Csv .\RushSongs.txt -Delimiter "`t"
2. Take a look at the parsed data:   
   $s|ft

# Task 4—Working with a Linux passwd file

A Linux passwd file is also a plaintext spreadsheet CSV file. It doesn’t have a header row, but PowerShell lets you specify a header row for a CSV file that doesn’t already have one.

## Steps

1. Now start a PowerShell Core (pwsh) session in Linux. The local accounts on every Linux system are in /etc/passwd, a *colon*-delimited data file. View its contents:   
   cat /etc/passwd
2. The passwd file does not have a row with column-headings, but the Import-Csv cmdlet lets us provide one with its named parameter -Header.
   1. Enter the following, all on one line:   
      $p = Import-Csv -Header "user","pw","uid","gid","GECOS","home","shell" -Delimeter ":" /etc/passwd
   2. Take a look at the parsed data:   
      $p|ft
   3. What is the UID (user ID) of your currently logged-in account? Click or tap here to enter text.
3. Most of the Linux local accounts are system service accounts, rather than actual user accounts. You can usually identify a service account because its shell is set to /usr/sbin/nologin or something similar. Insert a Foreach-Object filter into the pipeline to filter out all such accounts. (Modify the following command line as necessary for your selected Linux distribution.)   
   $p | % {if ($\_.shell -ne "/usr/sbin/nologin") {$\_}} | ft

# Deliverable

Upload this document with completed answers to I-Learn Canvas.