SOLID STATE DRIVE VALIDATION PROJECT

SCRIPTS

ImageDrive.sh

Stand-Alone Script

Parameters:

- Output folder
- Chosen index(es) of physical drives from list

The script's parameters can be previewed by passing the script a "?".

Ex) ./ImageDrive.sh ?

DESCRIPTION

Purpose: Creating an image of a drive.

Script will receive or prompt the user for an output file folder and then ask them to select a drive(s) from a **lsblk** list by its index. User may select any number of drives by their index (separated by commas) or select 0 to select all.

The script will only accept valid inputs, and it will continue to prompt user until valid input is received or is aborted by user. Script will also accept arguments passed to it and will skip the prompt(s) if the input is valid; otherwise, it will prompt the user for valid input.

Next, the script will create an image of each selected drive. The image file name will be generated by the script using the output folder, drive name, and current timestamp. If the image file already exists, the user will be prompted if they wish to overwrite and abort if no. Lastly, **dc3dd** is used to create an image in the selected output file folder with the following naming convention: DriveLabel-Year-Month-Day-Hour-Minute.dd. The user is notified when the output csv is ready.

Example image file name:

sdb-2019-06-27-09-27.dd

Sample script execution:

./ImageDrive.sh /data 1,2

./ImageDrive.sh /data 0

DriveValidation.sh

Is a Dependency (ExecutiveSummary.sh)

Has Dependency (OutputData.sh)

Parameters:

- Mount Folder
- Output File Folder
- Drive Type (Logical/Physical)
- If Physical, Chosen index(es) of physical drives from list
- If Logical, Image file path
- Partition Numbers

The script's parameters can be previewed by passing the script a "?".

Ex) ./DriveValidation.sh ?

DESCRIPTION

Purpose: Setting up the drive and its partitions for the creation of the file metadata files.

Script will receive or prompt the user for the mount folder, output file folder, Logical or Physical type, Devices, and Partitions. Mount folder is where the drive will be read-only mounted to. Output file folder is where the csv to be used for validation will be created. Device Type requires either an "L" for logical or "P" for physical. If device type is "L", user will provide disk image file paths separated by commas (.dd). If device type is "P", user will select physical devices from a **lsblk** list by index separated by commas. User will select partitions by index, separated by commas, from an **fdisk** list, and the user is provided with a prompt to retain the previous partition selection for the other selected devices.

Note: This script does not directly create the csv to be used for validation, rather it passes the information to its dependency script "OutputData.sh" so that it can create the csv. Due to having a dependency script, killing the script will have to be done twice to fully kill it.

The script will only accept valid inputs, and it will continue to prompt user until valid input is received or is aborted by user. Script will also accept arguments passed to it and will skip the prompt(s) if the input is valid; otherwise, it will prompt the user for valid input.

Next, the script will loop through each device and each partition. Each partition is then mounted as read-only. It will then generate a file name for the csv using the following naming convention: PartitionLabel-PartitionNumber-DriveType-Year-Month-Day-Hour-Minute.csv. Inputs and file name are passed to its dependency script "OutputData.sh" which will create the file metadata csv used for drive validation. The file metadata csv may then be compared to another version of the same drive to validate the drive.

Lastly, the script will wait for its dependency to finish before unmounting and repeating the process on the next partition if another one was selected by the user.

Note: This script will securely unmount drive before proceeding or it will prompt the user to abort if it was unsuccessful.

Example csv file name:

sdb-P1-TL-2019-06-27-09-27.csv

sdb-P1-TP-2019-06-27-09-22.csv

Sample script execution:

./DriveValidation.sh /mnt /data P 1,2 1

./DriveValidation.sh /mnt /data L /data/sdb-2019-06-31-07-25.dd,/data/sdb-2019-06-31-08-25.dd 1

OutputData.sh

Is a Dependency (DriveValidation.sh)

Parameters:

- Partition Name
- Mount Point
- Output file name

The script's parameters can be previewed by passing the script a "?".

Ex) ./OutputData.sh ?

DESCRIPTION

Purpose: Creating a File metadata csv used to display all possible ways a file could have been altered. This script's output is used by other scripts to show that one or more files have been altered; therefore, validating drive integrity.

Script will receive or prompt the user for a partition name, mount folder, and output file name. Partition name can be either a physical drive partition, such as /dev/sdb1, or an image drive partition, such as ./sdb-2019-06-27-09-27.dd1. Mount folder is the folder in which the partition is mounted to. Output file name will be the name of the csv.

The script will only accept valid inputs, and it will continue to prompt user until valid input is received or is aborted by user. Script will also accept arguments passed to it and will skip the prompt(s) if the input is valid; otherwise, it will prompt the user for valid input.

Note: If the csv file already exists, the user will be prompted if they wish to overwrite and abort if no.

Next, the file is then passed the file headers and a new entry is added for each file. The **find** command is used to loop through all files found and the **stat** command is used to get the file metadata. The **md5sum** command is used to get the hash of the file and hash of the metadata string. Each field is separated by a semicolon (;).

Important: Files with semicolons in their name will cause a single-field shift in the fields for that specific file. Manual edits to the csv will be required if this is the case. Files with this issue will present with one additional field when viewed as a spreadsheet.

Headers: File Name; File Hash; Inode Number; Access Rights; Blocks Allocated; File Type; Group Name; Size; User Name; Created Time; Access Time; Modification Time; Change Time; MetaData Hash.

Note: Metadata hash calculated using all fields with separators included.

Lastly, progress is displayed using **pv** so the user can see visual feedback of progress of adding an entry for each file, time spent, and estimated time left. The user is notified when the output csv is ready.

Access Rights, Owners, and Groups

Permissions regarding users and groups in NTFS-based file systems are handled differently than EXT-based file systems. Specifically, the stat command used to gather this information on EXT-based file systems does not accurately provide this information when applied to a NTFS file system. The current workaround for this is using **getfattr** to get creation time and access rights while using **ntfssecaudit** to get owner and group. OutputData.sh will switch to using this method when it detects an NTFS partition according to **fdisk -l**.

Creation Time

NTFS-based file systems store a creation time with its files while EXT-based file systems do not. The current work around for this is using **getfattr** which allows the script to get the creation time of the file. However, the timestamp does not have nanosecond precision as does the stat command for other timestamps. This is caused by having to convert the timestamp from Windows epoch to hex to Linux epoch then finally to the timestamp format. For display purposes, all timestamps will be recorded in the same format even though the creation time has no nanosecond precision.

Sample script execution:

./OutputData.sh /dev/sdb1 /mnt /data/sdb-P1-TP-2019-06-27-09-22.csv

ExecutiveSummary.sh

Has Dependency (DriveValidation.sh)

Parameters:

- Mount Point
- Output File Folder
- Drive Type (Logical/Physical)
- If Physical, Chosen index(es) of physical drives from list, else it takes an image file path

The script's parameters can be previewed by passing the script a "?".

Ex) ./ExecutiveSummary.sh ?

DESCRIPTION

Purpose: Creating a Summary csv used to show that one or more partitions on the drive has changed and that the entire drive is not 100% the same.

Script will receive or prompt the user for the mount folder, output file folder, Logical or Physical type, and Devices. Mount folder is the folder in which the partition is mounted to. Output file folder is the folder where the summary csv will be created. Device Type requires either an "L" for logical or "P" for physical. If device type is "L", user will provide single disk image file paths separated by commas (.dd). If device type is "P", user will select physical devices selected from a **lsblk** list by index separated by commas.

The script will only accept valid inputs, and it will continue to prompt user until valid input is received or is aborted by user. Script will also accept arguments passed to it and will skip the prompt(s) if the input is valid; otherwise, it will prompt the user for valid input.

Next, the script will create a temporary folder in the output directory and the folder will follow this naming convention: DriveLabel-Year-Month-Day-Hour-Minute. The script will send the required information to its dependency "DriveValidation.sh" and it will create a file metadata csv for each partition which will be used to create a summary for the whole drive.

Note: Due to having dependency scripts, killing the script will have to be done up to three times to fully kill it.

Next, the summary file is then passed the file headers and a new entry is added for every partition. Partition hash is generated using **md5sum** on the file metadata csv of the partition. Partition and drive information is gathered using **fdisk**. UUID and PARTUUID is gathered using **blkid**.

Headers: Partition Name; Validation Hash; Partition Size; Drive Name; Allocated Space; Sector Size; Disk Identifier; DiskLabel Type; UUID; PARTUUID

Note: UUID and PARTUUID are marked as (-) on filesystems that are not NTFS.

Lastly, the user is prompted if they wish to keep the temporary folder and validation csv files and will delete them if no. The user is notified when the summary csv is ready.

Sample script execution:

./ExecutiveSummary.sh /mnt /data P 2

./ExecutiveSummary.sh /mnt /data L /data/sdb-2019-06-27-09-27.dd

CombineCompare.sh

Requires Output Files (OutputData.sh)

Parameters:

- Output File Folder
- Original Validation CSV (Before)
- Current Validation CSV (After)

The script's parameters can be previewed by passing the script a "?".

Ex) ./CombineCompare.sh ?

DESCRIPTION

Purpose: Creating a Comparison csv used to show that one or more files on the before/after file metadata csv files has changed, been deleted, or been added.

Script will receive or prompt the user for the output file folder, original validation csv from "OutputData.sh", and the current validation csv from "OutputData.sh". Output File Folder is the folder where the comparison csv will be created. Original validation csv is the partition before. Current validation csv is the partition after.

The script will only accept valid inputs, and it will continue to prompt user until valid input is received or is aborted by user. Script will also accept arguments passed to it and will skip the prompt(s) if the input is valid; otherwise, it will prompt the user for valid input.

Next, the script will create a comparison file using the following naming convention: PartitionLabel-PartitionNumber-OF-Timestamp of Original File-CF-Timestamp of Current File-CC.csv.

Note: If the csv file already exists, the user will be prompted if they wish to overwrite and abort if no.

The script will output the headers then run through two different loops. First, it will compare the original to the current and append files that have been changed or deleted. Second, it will compare the current to the original and append files that have been added. Each file appended will be displayed on three rows.

Rows: Original; Current; Type

Note: This script uses **awk** to find matching entries on both files, so it may take time.

Type represents changed, deleted, added. Files that have changed will have a type "Changed" followed by a string of fields that have changed. Likewise, files that have been deleted will have a type "Deleted" and files that have been added will have a type "New".

Headers: Version; File Hash; File Name; Inode Number; Access Rights; Blocks Allocated; File Type; Group Name; Size; User Name; Created Time; Access Time; Modification Time; Change Time; MetaData Hash

Note: Files that have been deleted will not have anything under the "Current" row and files that have been added will not have anything under the "Original" row.

Lastly, the user is notified when the comparison csv is ready and is best analyzed as a spreadsheet.

Example csv file name:
sdb-P1-OF-2019-06-27-09-CF-2019-06-27-09-CC.csv

Sample script execution:
./CombineCompare.sh /data /data/sdb-P1-TL-2019-06-27-09-27.csv /data/sdb-P1-TP-2019-06-27-09-28.csv

GenDataLoop.sh

Stand-Alone Script

Parameters:

- Mount folder
- Chosen index(es) of physical drives from list
- Chosen index(es) of partition(s) from list

The script's parameters can be previewed by passing the script a "?".

Ex) ./GenDataLoop.sh ?

DESCRIPTION

Purpose: Creating test data, files, and directory structures to run the other scripts against.

Script will receive or prompt the user for a mount folder, drives, and partitions. Mount folder will be where the partition will be read-only mounted to. User will select a physical device from a **lsblk** list by its index. User will select partition indexes for each selected drive from a **fdisk** list with a prompt to retain previously selected partitions when the script moves on to another selected drive.

The script will only accept valid inputs, and it will continue to prompt user until valid input is received or is aborted by user. Script will also accept arguments passed to it and will skip the prompt(s) if the input is valid; otherwise, it will prompt the user for valid input.

Note: This script is for purposes of setting up tests by creating test data to analyze the effects of the other scripts.

Next, the script will mount the partition, delete all pre-existing files on it, and create randomly generated files and folders to simulate a drive's contents. File contents are randomized by appending a pre-defined lorem ipsum string a random number of times.

Note: This does sometimes result in files having the same file hash but having a different metadata hash.

Files will receive a random name, data, permissions, owner, group, and extension. Files will continue to be generated till the partition is almost full. Folders are randomly created to simulate directory trees.

Note: This script will securely unmount drive before proceeding or it will prompt the user to abort if it was unsuccessful.

Sample script execution:

./GenDataLoop.sh /mnt 2 1

AlterFiles.sh

Stand-Alone Script

Parameters:

- Partition Name
- Mount folder

The script's parameters can be previewed by passing the script a "?".

Ex) ./AlterFiles.sh ?

DESCRIPTION

Purpose: Simulating file alterations that a user could make while making a 'best attempt' to isolate these alterations to modify one metadata field at a time.

Script will receive or prompt the user for partition name and mount folder. Partition name can be either a physical drive partition, such as /dev/sdb1, or an image drive partition, such as ./sdb-2019-06-27-09-27.dd1. Mount folder is the folder in which the partition is mounted to.

The script will only accept valid inputs, and it will continue to prompt user until valid input is received or is aborted by user. Script will also accept arguments passed to it and will skip the prompt(s) if the input is valid; otherwise, it will prompt the user for valid input.

Note: This script is for purposes of setting up tests to verify that the metadata hash responds as expected to file changes.

Next, the script will randomly select files on the partition and will attempt to make one change to its metadata. This script will attempt to alter the file type, permissions, access time, group, owner, size, modification time, and change time respectively for the selected files. It is not possible for this script to isolate a change for certain fields due to the nature of its interactions with other fields. However, it succeeded in verifying that the metadata hash does reflect that metadata has changed on a file and can with confidence show that the partition has been altered in some way.

Note: This script will securely unmount drive before proceeding or it will prompt the user to abort if it was unsuccessful.

Sample script execution:

./AlterFiles.sh /dev/sdb1 /mnt

./AlterFiles.sh /data/sdb-2019-06-27-09-27.dd1 /mnt

OutputData.ps1

Discontinued Script

Parameters:

- Partition Name
- Mount folder

The script's parameters can be previewed by passing the script a "?".

Ex) ./OutputData.ps1 ?

DESCRIPTION

Purpose: Creating a File metadata csv used to display all possible ways a file could have been altered. This is a Windows-specific solution that was discontinued.

Script will receive or prompt the user for a folder and output file name. It would find a specific file or all files within the folder and return a string of its metadata, file hash, and metadata hash.

The script will only accept valid inputs, and it will continue to prompt user until valid input is received or is aborted by user. Script will also accept arguments passed to it and will skip the prompt(s) if the input is valid; otherwise, it will prompt the user for valid input.

Note: If the csv file already exists, the user will be prompted if they wish to overwrite and abort if no.

Next, the file is then passed the file headers and a new entry is added for each file.

Note: Script could not be ported to Linux filesystems through PowerShell Core due to limitations of not having a module for **Acl** or **fsutil**. Script can only function on windows filesystems and was discontinued.

Lastly, the script would display a progress visual for users to track the progress of the script, time elapsed, and estimated time remaining. The user is notified when the output csv is ready.