

```
File Actions Edit View Help
(kali@kali)-[~]
$ sudo fdisk -l
[sudo] password for kali:
Disk /dev/sda: 80.09 GiB, 86000000000 bytes, 167968750 sectors
Disk model: VMware Virtual S
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xa2cae31e

Device      Boot Start      End  Sectors  Size Id Type
/dev/sda1   *    2048 167968749 167966702  80.1G 83 Linux
```

This screenshot shows the use of the `fdisk -l` command on a Kali Linux terminal. It displays the partition details of `/dev/sda`, which is an 80.09 GiB disk. It shows sector sizes of 512 bytes, both logically and physically. The partition table is of the dos type, with the disk's

identifier. There's one partition (`/dev/sda1`), marked as bootable, starting at sector 2048 and ending at 167968749, with a size of 80.1 GiB and a Linux file system (Id 83).

```
(kali@kali)-[~]
$ cd /dev
(kali@kali)-[/dev]
$ ls
autofs      disk        hpet        midi         pts         snd          tty13      tty22      tty31      tty40      tty5         tty59      ttyS1      vcs3      vcsa5      vcsu7
block       dmideid     hugepages   queue        random      stderr       tty14      tty23      tty32      tty41      tty6         ttyS2      vcs4      vcsa6      vfuio
bsg         dri         hwrng       net          rkill       stdin        tty15      tty24      tty33      tty42      tty51      ttyS3      vcs5      vcsa7      vga_arbiter
brfs-control fb0         initctl     null         rtc          stdout       tty16      tty25      tty34      tty43      tty52      tty61      uhid       vcs6      vcsu       vhci
bus         fd          input       nvram        rtc0        tty          tty17      tty26      tty35      tty44      tty53      tty62      uinput     vcs7      vcsu1      vhost-net
char        full        kmsg        parport0     sda         tty0         tty18      tty27      tty36      tty45      tty54      tty63      urandom    vcsa      vcsu2      vhost-vsock
console     fuse        log         port         sda1        tty1         tty19      tty28      tty37      tty46      tty55      tty7       userfaultfd vcsa1      vcsu3      vmci
core        hidraw0     loop-control ppp          sg0         tty10        tty20      tty29      tty38      tty47      tty56      tty8       vcs       vcsa2      vcsu4      vsock
cpu_dma_latency hidraw1     mapper      psaux        shm         tty11        tty21      tty30      tty48      tty57      tty9       vcs1       vcsa3      vcsu5      zero
cuse        hidraw2     mem         ptmx         snapshot    tty12        tty21      tty30      tty49      tty58      tty50      vcs2       vcsa4      vcsu6
```

This screenshot shows the Kali Linux terminal after navigating to the `/dev` directory and listing its contents using the `ls` command. The `/dev` directory holds device files, representing various hardware components and devices recognized by the Linux system. It shows a range of device files including `sda`, `sda1`, and `sg0`, which correspond to disk and partition devices, as well as other system devices like `tty`, `null`, `random`, and more.

```
(kali@kali)-[~]
$ sudo apt install lshw
Installing:
lshw

Summary:
  Upgrading: 0, Installing: 1, Removing: 0, Not Upgrading: 911
  Download size: 304 kB
  Space needed: 958 kB / 64.4 GB available

Get:1 http://http.kali.org/kali kali-rolling/main amd64 lshw amd64 02.19.git.2021.06.19.996aaad9c7-2+b2 [304 kB]
Fetched 304 kB in 1s (588 kB/s)
Selecting previously unselected package lshw.
(Reading database ... 395913 files and directories currently installed.)
Preparing to unpack .../lshw_02.19.git.2021.06.19.996aaad9c7-2+b2_amd64.deb ...
Unpacking lshw (02.19.git.2021.06.19.996aaad9c7-2+b2) ...
Setting up lshw (02.19.git.2021.06.19.996aaad9c7-2+b2) ...
Processing triggers for kali-menu (2024.3.1) ...
Processing triggers for man-db (2.12.1-2) ...
```

The `sudo apt install lshw` command is being executed in the terminal to install the `lshw` tool, which is used to list hardware components of the system. The output shows the process of installing `lshw`, including fetching package information, unpacking, and setting it up. The installation is successful, with a summary indicating no errors.

```
(kali㉿kali)-[~]
└─$ sudo lshw -class disk -short
H/W path          Device          Class          Description
-----
/0/100/10/0.0.0   /dev/sda        disk           86GB VMware Virtual S
```

The `sudo lshw -class disk -short` command is used to generate a concise overview of the disk devices recognized by the system. The output lists one disk under the hardware path `/0/100/10/0.0.0`, identified as `/dev/sda`. This disk is classified as `disk` with a description showing its capacity of 86GB, further indicating the virtualized environment. The output provides a quick summary of the storage hardware without delving into specific partitions or volumes, which is useful for verifying the physical or virtual disks connected to the system.

```
(kali㉿kali)-[~]
└─$ printf cs362 | sha1sum
ee337f581bdf94a9270c7d6ac33acb58659d40a2 -

(kali㉿kali)-[~]
└─$ printf cs362 | md5sum
21e807599f8ec807297d3f9d9bcb635 -

(kali㉿kali)-[~]
└─$ printf cs362 | sha512sum
be47fe03860b2c7330b2d15bb7911fbd4b5e73327b35d1a1857537948f92f92f3aaf28fb56bc595d5d8f0a9fdf580fb294840f33a2df3c4fd46f07cc2cfe9bd97 -
```

This screenshot displays the use of three different hashing commands in Linux: `sha1sum`, `md5sum`, and `sha512sum`. The command `printf cs362` outputs the string `cs362`, which is then piped into each hashing command to generate a unique hash for the input string. The output shows three different hash values: a 160-bit SHA-1 hash, a 128-bit MD5 hash, and a 512-bit SHA-512 hash, which is a much longer hexadecimal string. This exercise illustrates how various hashing algorithms produce unique digests for a given input, showing the different hash lengths and complexity, which play an important role in digital forensics for verifying data integrity.

```
(kali㉿kali)-[~]
└─$ echo this is a text file > file1.txt

(kali㉿kali)-[~]
└─$ md5sum file1.txt
fda4e701258ba56f465e3636e60d36ec  file1.txt
```

This shows the creation of a simple text file named `file1.txt` using the `echo` command with the content "this is a text file." The next command, `md5sum file1.txt`, generates an MD5 hash for this file, resulting in a hash. This

process demonstrates how to create a file and compute its hash, a common practice in digital forensics to verify that the contents of a file have not been altered. The MD5 hash acts as a digital fingerprint for the file's current state.

```
(kali㉿kali)-[~]
└─$ md5sum Downloads/*
160479a83f255cc01e57ea62f88104d9  Downloads/Flower.jpeg
be1fb407f111fa554799707194966f9b  Downloads/Nature.jpeg
```

The `md5sum` command is used to calculate the MD5 hashes for all files in the Downloads directory, indicated by `Downloads/*`. Two JPEG files, `Flower.jpeg` and

`Nature.jpeg`, have their hash values displayed. The purpose of generating these hashes is to ensure the integrity of the files, which is especially useful when monitoring files for any unauthorized changes. This method provides a quick way to hash all files within a specific directory.

```
(kali@kali)-[~]
$ printf cs362 | openssl dgst -sha3-256
SHA3-256(stdin)= e4ca8e0e958b39280f5ba86cd8864b194645c37ac1b89a778416a1bf23e4ef0a

(kali@kali)-[~]
$ openssl dgst -sha3-256 Downloads/*
SHA3-256(Downloads/Flower.jpeg)= 3c4c330f553ed9b070bfb57b6be23f983a318d5312a357ad90c072d1412ed4a5
SHA3-256(Downloads/Nature.jpeg)= a5009b02f694ae783929e246fa7b259a25ceb1fafbb998fbee49b8bf847c141
```

This screenshot shows the use of the openssl command to generate SHA3-256 hashes for a string and for files in the Downloads directory. The command `printf cs362 | openssl dgst -sha3-256` outputs the SHA3-256 hash for the string cs362. The command calculates the SHA3-256 hashes for Flower.jpeg and Nature.jpeg, resulting in two unique hash values. SHA-3 is part of a newer family of cryptographic hash functions, offering enhanced security compared to previous SHA versions. This demonstrates how to use OpenSSL for advanced hashing operations.

```
(kali@kali)-[/]
$ sudo dc3dd if=/dev/sdb hash=sha1 log=usb_forensics.log

dc3dd 7.2.646 started at 2024-09-25 12:55:15 -0400
compiled options:
command line dc3dd if=/dev/sdb hash=sha1 log=usb_forensics.log
device size: 4194304 sectors (probed), 2,147,483,648 bytes
sector size: 512 bytes (probed)
2147483648 bytes ( 2 G ) copied ( 100% ), 89 s, 23 M/s

input results for device `/dev/sdb':
4194304 sectors in
0 bad sectors replaced by zeros
91d50642dd930e9542c39d36f0516d45f4e1af0d (sha1)

output results for file `stdout':
4194304 sectors out

dc3dd completed at 2024-09-25 12:56:43 -0400
```

This shows us the use of the dc3dd command to create a forensic image of the /dev/sdb drive. The command specifies the input file (/dev/sdb), the hashing algorithm (SHA-1), and the log file (usb\_forensics.log). The output shows that the command successfully copied 2 GiB of data with a sector size of 512 bytes, generating a SHA-1 hash. The log captures details about the imaging process, including

the number of sectors read (4,194,304) and confirms that there were no bad sectors. This process is essential in digital forensics to ensure the integrity of the copied data and to create a bit-by-bit copy of the disk for further analysis. The final line indicates the completion of the imaging process, which took about 89 seconds.

```
(kali@kali)-[/]
$ sudo dc3dd if=/dev/sdb hash=sha1 of=usb_image.dd

dc3dd 7.2.646 started at 2024-09-25 12:58:40 -0400
compiled options:
command line dc3dd if=/dev/sdb hash=sha1 of=usb_image.dd
device size: 4194304 sectors (probed), 2,147,483,648 bytes
sector size: 512 bytes (probed)
2147483648 bytes ( 2 G ) copied ( 100% ), 19 s, 108 M/s

input results for device `/dev/sdb':
4194304 sectors in
0 bad sectors replaced by zeros
91d50642dd930e9542c39d36f0516d45f4e1af0d (sha1)

output results for file `usb_image.dd':
4194304 sectors out

dc3dd completed at 2024-09-25 12:58:59 -0400
```

This screenshot shows the use of the dc3dd command to create a forensic image of the /dev/sdb drive and save it as usb\_image.dd. The tool reads the drive, generating a SHA-1 hash to ensure data integrity. It successfully copied 2 GiB of data in 19 seconds at a speed of 108 MB/s, confirming that there were no bad sectors. The operation completed successfully, creating a bit-by-bit copy of the drive for forensic analysis.

```
(kali@kali)-[/]
$ ls
bin  dev  home  initrd.img.old  lib32  lost+found  mnt  proc  run  srv  sys  usb_forensics.log  usr  vmlinuz
boot  etc  initrd.img  lib  lib64  media  opt  root  sbin  swapfile  tmp  usb_image.dd  var  vmlinuz.old
```

```
(kali@kali)-[/]
$ ls -l
total 7340128
lrwxrwxrwx 1 root root 7 Aug 12 10:24 bin -> usr/bin
drwxr-xr-x 3 root root 4096 Aug 18 18:30 boot
drwxr-xr-x 17 root root 3280 Sep 25 12:47 dev
drwxr-xr-x 185 root root 12288 Sep 25 12:51 etc
drwxr-xr-x 3 root root 4096 Aug 18 15:57 home
lrwxrwxrwx 1 root root 28 Aug 18 18:29 initrd.img -> boot/initrd.img-6.8.11-amd64
lrwxrwxrwx 1 root root 28 Aug 18 18:29 initrd.img.old -> boot/initrd.img-6.8.11-amd64
lrwxrwxrwx 1 root root 7 Aug 12 10:24 lib -> usr/lib
lrwxrwxrwx 1 root root 9 Aug 18 15:49 lib32 -> usr/lib32
lrwxrwxrwx 1 root root 9 Aug 12 10:24 lib64 -> usr/lib64
drwx----- 2 root root 16384 Aug 18 18:25 lost+found
drwxr-xr-x 2 root root 4096 Aug 18 15:37 media
drwxr-xr-x 2 root root 4096 Aug 18 15:37 mnt
drwxr-xr-x 3 root root 4096 Aug 18 15:49 opt
dr-xr-xr-x 289 root root 0 Sep 25 12:43 proc
drwx----- 4 root root 4096 Sep 14 21:22 run
drwxr-xr-x 36 root root 880 Sep 25 12:43 sbin
drwxr-xr-x 3 root root 4096 Aug 18 15:52 srv
-rw----- 1 root root 1073741824 Aug 18 18:29 swapfile
dr-xr-xr-x 13 root root 0 Sep 25 12:43 sys
drwxrwxrwt 15 root root 340 Sep 25 12:52 tmp
-rw-r--r-- 1 root root 576716800 Sep 25 13:06 usb_forensics.000
-rw-r--r-- 1 root root 576716800 Sep 25 13:06 usb_forensics.001
-rw-r--r-- 1 root root 576716800 Sep 25 13:06 usb_forensics.002
-rw-r--r-- 1 root root 417333248 Sep 25 13:06 usb_forensics.003
-rw-r--r-- 1 root root 2147483648 Sep 25 13:03 usb_forensics.info
-rw-r--r-- 1 root root 540 Sep 25 12:56 usb_forensics.log
-rw-r--r-- 1 root root 2147483648 Sep 25 12:58 usb_image.dd
drwxr-xr-x 16 root root 4096 Aug 18 15:49 usr
drwxr-xr-x 12 root root 4096 Sep 14 21:21 var
lrwxrwxrwx 1 root root 25 Aug 18 18:29 vmlinuz -> boot/vmlinuz-6.8.11-amd64
lrwxrwxrwx 1 root root 25 Aug 18 18:29 vmlinuz.old -> boot/vmlinuz-6.8.11-amd64
```

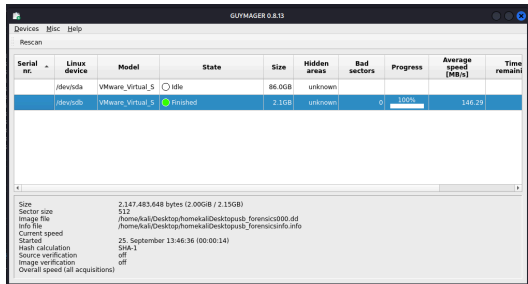
This screenshot shows the output of the `ls -l` command in the root directory (`/`), providing a detailed list of files and directories along with their permissions, ownership, sizes, and modification dates. Standard system directories like `bin`, `boot`, `dev`, and `etc` are present, each with root ownership. Here are files named `usb_forensics.000`, `usb_forensics.001`, `usb_forensics.002`, `usb_forensics.003`, and `usb_forensics.info`, suggesting the forensic image was either split into parts or accompanied by additional metadata files. This listing provides an overview of the filesystem's current state, highlighting the forensic files recently added or modified.

```
(kali@kali)-[/]
$ sudo dd if=/dev/sdb of=/dev/sdb
dd300 7.2.646 started at 2024-09-25 13:26:25 -0400
compiled options:
command line dd300 wiper=/dev/sdb
device size: 4194384 sectors (probed), 2,147,483,648 bytes
sector size: 512 bytes (probed)
2147483648 bytes (2.0 GiB) copied (100%), 5 s, 445 M/s
input results for pattern '00':
4194384 sectors in
output results for device '/dev/sdb':
4194384 sectors out
dd300 completed at 2024-09-25 13:26:29 -0400

(kali@kali)-[/]
$ sudo dd if=/dev/sdb of=/dev/sdb bs=512 count=1
dd300 7.2.646 started at 2024-09-25 13:27:13 -0400
compiled options:
command line dd300 wiper=/dev/sdb bs=512 count=1
device size: 4194384 sectors (probed), 2,147,483,648 bytes
sector size: 512 bytes (probed)
2147483648 bytes (2.0 GiB) copied (100%), 6 s, 358 M/s
input results for pattern 'ABCDEF':
4194384 sectors in
output results for device '/dev/sdb':
4194384 sectors out
dd300 completed at 2024-09-25 13:27:19 -0400

(kali@kali)-[/]
$ sudo dd if=/dev/sdb of=/dev/sdb bs=512 count=1
dd300 7.2.646 started at 2024-09-25 13:27:56 -0400
compiled options:
command line dd300 wiper=/dev/sdb bs=512 count=1
device size: 4194384 sectors (probed), 2,147,483,648 bytes
sector size: 512 bytes (probed)
2147483648 bytes (2.0 GiB) copied (100%), 5 s, 393 M/s
input results for pattern 'happyholidays':
4194384 sectors in
output results for device '/dev/sdb':
4194384 sectors out
dd300 completed at 2024-09-25 13:28:01 -0400
```

This step shows the use of the `dc3dd` tool to wipe the contents of the `/dev/sdb` drive with different patterns. The first command overwrites the drive with zeros (`00`), processing 4,194,304 sectors at a speed of 445 MB/s, completing in 5 seconds. The second command uses the pattern "ABCDEF" for wiping, completing in 6 seconds at 358 MB/s. The third command uses the custom text pattern "happyholidays," finishing in 5 seconds at 393 MB/s. Each operation confirms that all sectors were successfully overwritten, ensuring that the data on the drive was securely erased.



```
(kali@kali)-[~]
$ sudo dd if=/dev/sdb bs=512 of=mbr.image count=1
1+0 records in
1+0 records out
512 bytes copied, 0.00134819 s, 380 kB/s
```

The Screenshot on the left displays the interface of Guymager, a forensic disk imaging tool. It shows two devices: `/dev/sda` and `/dev/sdb`. The imaging process for `/dev/sdb` has finished, as indicated by the progress bar and status. The details at the bottom reveal that the disk was successfully imaged with a sector size of 512 bytes and a total size of 2,147,483,648 bytes. The average speed during the process was 146.29 MB/s, and SHA-1 hashing was used for data integrity. The resulting image file is saved, along with an associated info file. This overview confirms that the disk imaging

process was completed successfully and securely. The screenshot on the right shows the use of the dd command to create an image of the Master Boot Record (MBR) of the /dev/sdb drive. The command `sudo dd if=/dev/sdb bs=512 of=mbr.image count=1` copies the first 512 bytes (which includes the MBR) from the drive into a file named mbr.image. The output indicates that 512 bytes were successfully copied in a fraction of a second at a speed of 380 kB/s. This operation is commonly used in disk forensics to extract the MBR for analysis.