An-Najah National University

Faculty of Engineering and Information Technology

Network Administration Lab

Experiment#2

Dr. Bakr Abd Al-Haq

Team Members: Ali Khuffash, Wajd Abd Al-Hadee

Overview

This experiment, conducted in the Network Administration Lab at An-Najah National University, focuses on essential file system management tasks for network administrators. It covers topics such as disk partitioning, file system types, mounting, and file permissions in Linux. The experiment involves hands-on activities like mounting external file systems, managing soft and hard links, and automating tasks through scripting. Additionally, the sticky bit concept is explored, ensuring file security in shared directories. Challenges faced during the experiment, such as mounting Windows file systems in a virtual environment, are also discussed along with the solutions implemented.

Procedure & Analysis

3.1 File System Partitions

A hard disk is divided into partitions, and each partition can be assigned a different file system depending on the operating system, such as NTFS, FAT32, and exFAT on Windows, EXT4, XFS, and Btrfs on Linux, and APFS and HFS+ on macOS. Other file systems like ReFS and ZFS are also used. Some operating systems support reading and writing multiple file system types. The fdisk -l command was used to display all the partitions on the machine. The output shows the details of the partitions, including the one assigned with the Linux file system, typically labeled as Linux or Linux filesystem. See figure 3.1.1.

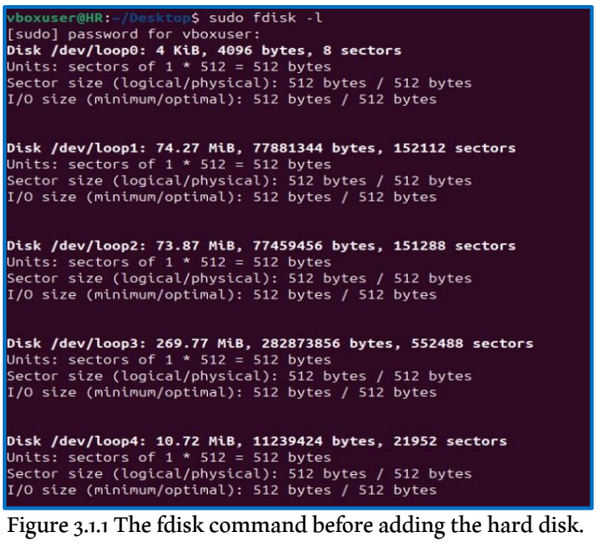

Figure 3.1.1 The fdisk command before adding the hard disk.

The /proc directory uses the proc file system, a virtual file system that provides information about system processes and hardware. The mount command can be used

to check mounted partitions and their locations. See figure 3.1.2

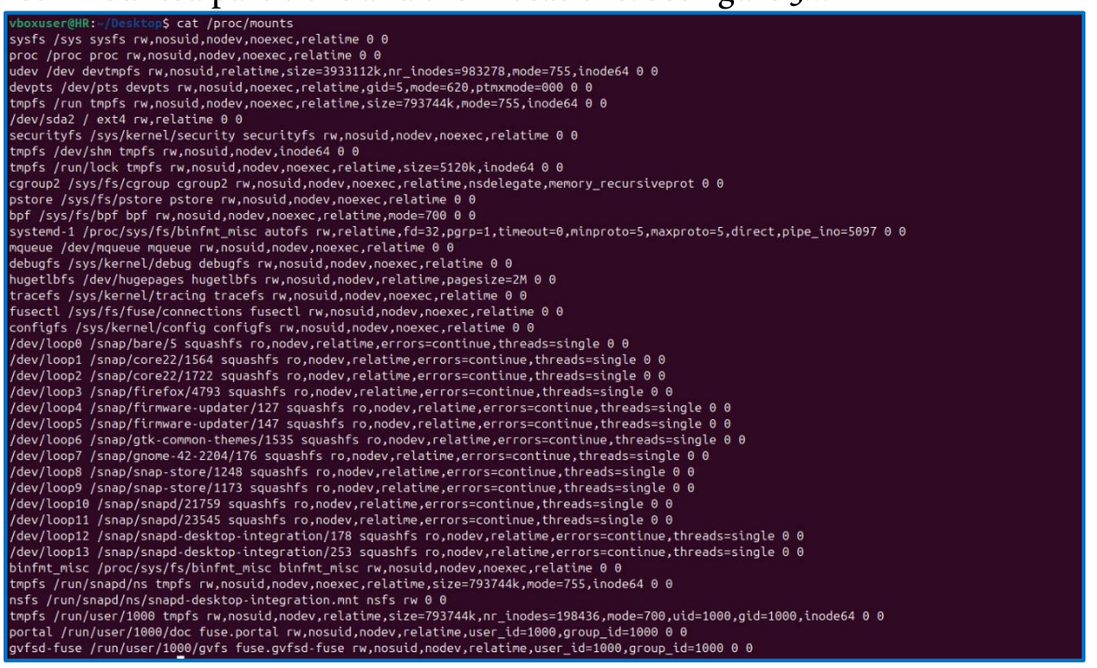

Figure 3.1.2 The mount command to display mounted files.

You can check the file systems supported by your Linux system using /proc/filesystems, which lists the available options. File systems marked with nodev are virtual and do not require a physical storage device. See figure 3.1.3

Figure 3.1.3 Listing available filesystems using the /proc/filesystems file.

When booting Linux, a set of file systems are automatically mounted. You can check the list of automatically mounted files by viewing /etc/fstab. Each line in this file includes:

- **Device**: Specifies the partition or UUID.
- **Mount Point**: The location where it is mounted.
- **File System Type**: Such as ext4 or xfs.
- **Mount Options**: Like defaults or ro.
- **Dump Option**: Used for backups, usually set to 0.

- **Fsck Order**: Defines the file system check priority (0 = no check, 1 = root, 2 = others). See figure 3.1.4

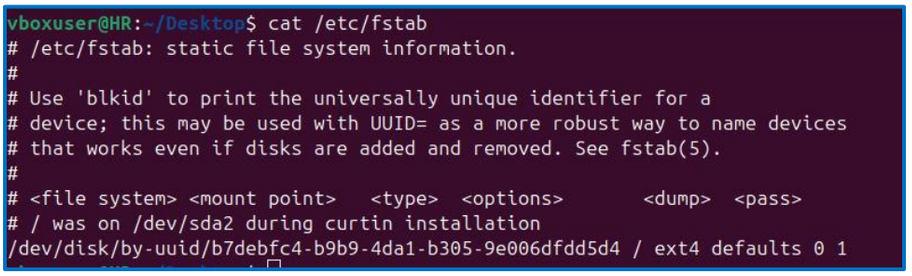

Figure 3.1.4 The information inside the /etc/fstab file.

To mount the Windows file system on Linux, first, identify the partition containing the NTFS or FAT32 file system using commands like lsblk -f or fdisk -l. Then, add the line /dev/XYZ /mnt/win vfat noauto 0 0 to the /etc/fstab file, where vfat represents the FAT32 file system, and noauto prevents automatic mounting at boot. See figure 3.1.4

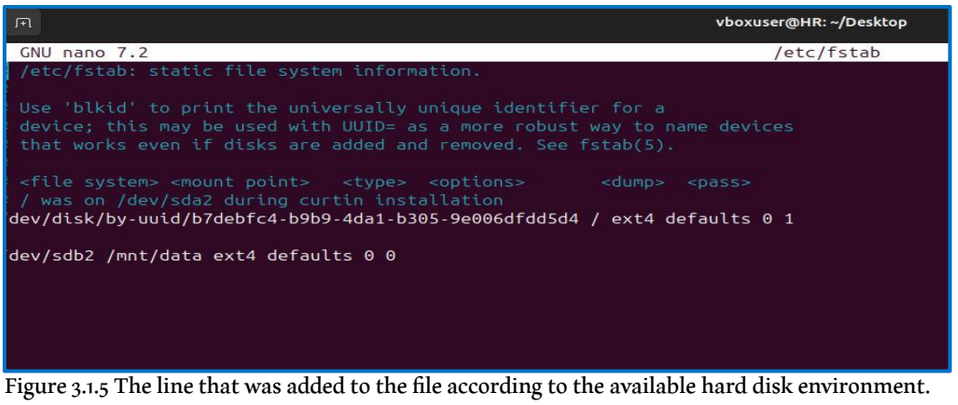

Figure 3.1.5 The line that was added to the file according to the available hard disk environment.

To create the /mnt/win directory, use the command mkdir /mnt/win. Root permissions are required because /mnt is an important directory in the file system and requires elevated permissions for modification. See figure 3.1.6

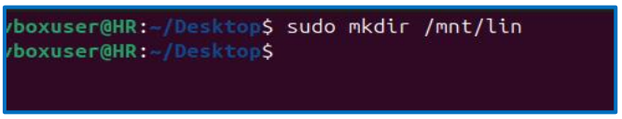

Figure 3.1.6 Creating the directory by using sudo permeation.

The mounting process by using mount command. See figure 3.1.7 and figure 3.1.8

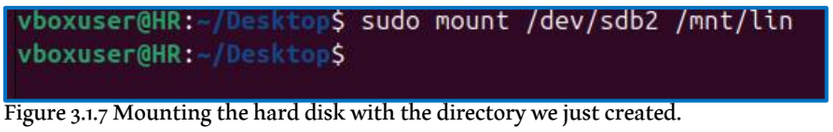

Figure 3.1.7 Mounting the hard disk with the directory we just created.

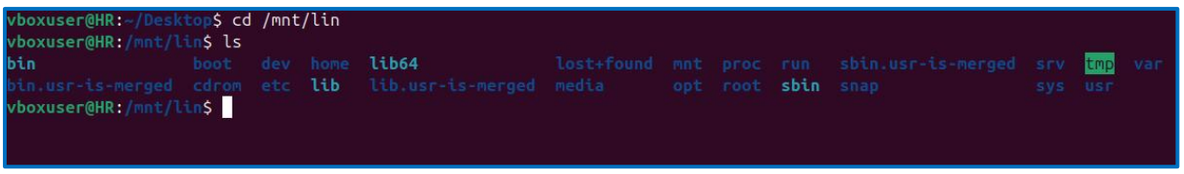

Figure 3.1.8 Displaying the files that have been mounted in the mnt/lin directory.

3.2 File Types

The types of files in the Linux file system are:

- Regular file (-)
- Directory (d)
- Symbolic link (l)
- Character device file (c)
- Block device file (b)
- FIFO (Named pipe) (p)
- Socket file (s)

See figure 3.2.1 and figure 3.2.2

```
ali@ali-VirtualBox:~/Desktop$ mkdir Exp5
ali@ali-VirtualBox:~/Desktop$ cd Exp5
ali@ali-VirtualBox:~/Desktop/Exp5$ touch file1.txt
```

Figure 3.2.1 Created a directory named 'Exp5' and added a file named 'file1.txt' inside it.

```
ali@ali-pc:~/Desktop$ mkdir Exp5
ali@ali-pc:~/Desktop$ touch file1.txt
ali@ali-pc:~/Desktop$ realpath "file1.txt"
/home/ali/Desktop/file1.txt
ali@ali-pc:~/Desktop$
```

Figure 3.2.2 absolute path to file1.txt.

The files and folders under the /dev directory include:

- Regular file: Example: /dev/null
- Directory: Example: /dev/pts
- Character device file: Example: /dev/tty
- Block device file: Example: /dev/sda
- Symbolic link: Example: /dev/cdrom

See figure 3.2.3 and figure 3.2.4

```
ali@ali-VirtualBox:~/Desktop/Exp5$ ls -l /dev
total 0
crw-r--r--  1 root    root     10, 235 Feb 20 11:03 autofs
drwxr-xr-x  2 root    root          400 Feb 20 11:03 block
drwxr-xr-x  2 root    root           80 Feb 20 11:03 bsg
crw-rw----  1 root    disk     10, 234 Feb 20 11:03 btrfs-control
drwxr-xr-x  3 root    root           60 Feb 20 11:03 bus
lrwxrwxrwx  1 root    root            3 Feb 20 11:03 cdrom -> sr0
drwxr-xr-x  2 root    root        3720 Feb 20 11:03 char
crw--w----  1 root    tty         5,   1 Feb 20 11:03 console
lrwxrwxrwx  1 root    root           11 Feb 20 11:03 core -> /proc/kcore
drwxr-xr-x  4 root    root           80 Feb 20 11:03 cpu
crw-------  1 root    root     10, 124 Feb 20 11:03 cpu_dma_latency
crw-------  1 root    root     10, 203 Feb 20 11:03 cuse
drwxr-xr-x  6 root    root          120 Feb 20 11:03 disk
drwxr-xr-x  2 root    root           60 Feb 20 11:03 dma_heap
drwxr-xr-x  3 root    root          100 Feb 20 11:03 dri
lrwxrwxrwx  1 root    root            3 Feb 20 11:03 dvd -> sr0
crw-------  1 root    root     10, 126 Feb 20 11:03 ecryptfs
```

Figure 3.2.3 Listing device files in the /dev directory.

```
ali@ali-VirtualBox:~/Desktop/Exp5$ ls -l /dev |grep  "null"
crw-rw-rw-  1 root    root       1,   3 Feb 20 11:03 null
```

Figure 3.2.4 Filtering a specific device file using grep in /dev.

The difference between a block device file (b) and a character device file (c):

- Block Device (b): Transfers data in fixed-size blocks, used for storage devices like hard drives (/dev/sda).
- Character Device (c): Transfers data as a stream, used for devices like keyboards and serial ports (/dev/tty)

The script checks if the directory provided as an argument exists. If it exists, it finds all symbolic links within the directory and writes their names to a file named Links.txt. If the directory does not exist, it displays an error message and stops execution. See figure 3.2.5

```
#!/bin/bash


DESKTOP_PATH="$HOME/Desktop"


if [ -z "$1" ]; then
    echo "1111."
    exit 1
fi


if [ ! -d "$1" ]; then
    echo "the desktop doesn't exists!"
    exit 1
fi


LINKS_FILE="$DESKTOP_PATH/Links.txt"
touch "$LINKS_FILE"


find "$1" -type l > "$LINKS_FILE"


echo "We save all the  symbolic link in: $LINKS_FILE"
```

Figure 3.2.5 Bash script to find and save symbolic links.

To install the socket package or the relevant tools (such as netcat) for working with sockets, use the appropriate package manager for your distribution. See figure 3.2.6

```
ali@ali-VirtualBox:~/Desktop$ sudo apt install socat -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libfprint-2-tod1 libllvm10 linux-image-5.4.0-42-generic
  linux-modules-5.4.0-42-generic linux-modules-extra-5.4.0-42-generic
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  socat
0 upgraded, 1 newly installed, 0 to remove and 130 not upgraded.
Need to get 323 kB of archives.
After this operation, 1,394 kB of additional disk space will be used.
Get:1 http://ps.archive.ubuntu.com/ubuntu focal/main amd64 socat amd64 1.7.3.3-2
 [323 kB]
Fetched 323 kB in 1s (608 kB/s)
Selecting previously unselected package socat.
(Reading database ... 227224 files and directories currently installed.)
Preparing to unpack .../socat_1.7.3.3-2_amd64.deb ...
Unpacking socat (1.7.3.3-2) ...
Setting up socat (1.7.3.3-2) ...
Processing triggers for man-db (2.9.1-1) ...
```

Figure 3.2.6 Installing the socat package using apt in Linux.

To create a UNIX domain stream socket, you specify the socket type as AF_UNIX and SOCK_STREAM, using a file path (e.g., /tmp/socketfile) instead of a port number. UNIX domain sockets do not use port numbers like TCP/IP sockets. See figure 3.2.7

Figure 3.2.7 Creating a TCP socket using socat.

For checking all listening sockets and make sure that your socket created socket is there. See figure 3.2.8

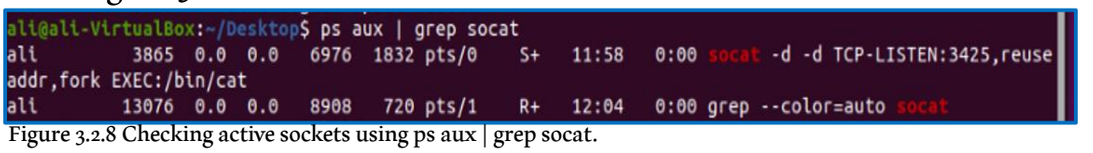

Figure 3.2.8 Checking active sockets using ps aux | grep socat.

For killing the running socket process, see figure 3.2.9

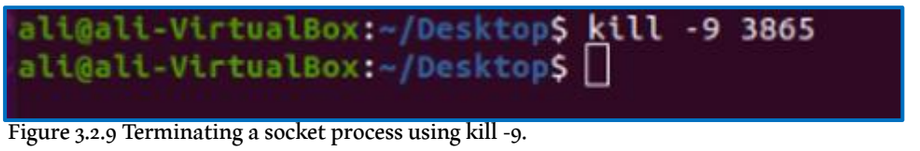

Figure 3.2.9 Terminating a socket process using kill -9.

3.3 File Symbolic Links

The key difference between a soft link (symbolic link) and a hard link is:

Soft link: Points to the name file. Breaks if the original file is deleted. Created with ln -s.

Hard link: Points directly to the file's data (i-node). Remains valid even if the original file is deleted, as the data is still accessible through the hard link. Created with ln.

In short, a soft link points to the file path, while a hard link points to the file's data. See from figure 3.3.1 to figure 3.3.3

Figure 3.3.1 Touching two files, the first named f1 and the second named f2.

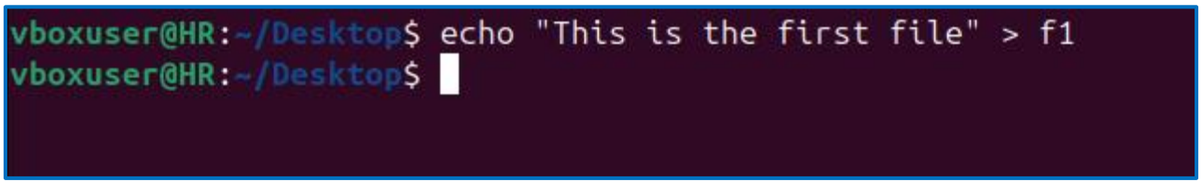

Figure 3.3.2 Write "This is the first file" in file f1.

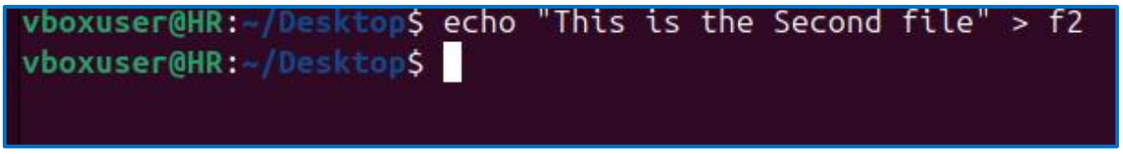

Figure 3.3.3 Write "This is the Second file" in file f2.

Creating the hard link and soft link for files. See figure 3.3.4 and figure 3.3.5

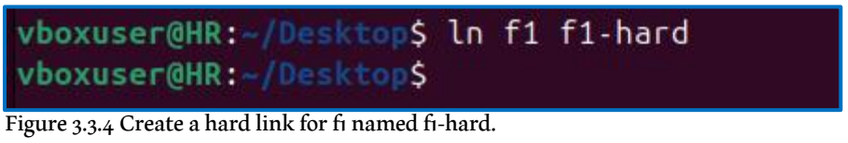

Figure 3.3.4 Create a hard link for f1 named f1-hard.

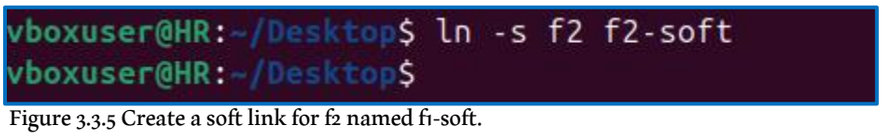

Figure 3.3.5 Create a soft link for f2 named f1-soft.

Renaming the originals files, see figure 3.3.6 and figure 3.3.7

Figure 3.3.6 Rename f1 to f1-new.

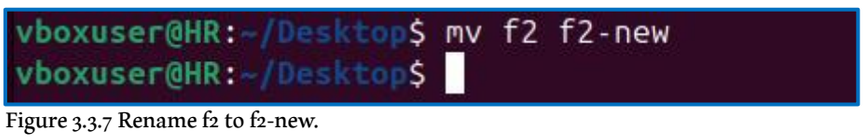

Figure 3.3.7 Rename f2 to f2-new.

When printing the content of each file, we will find that the content of the soft link has disappeared, as it now points to null after the file name change. However, the content of the hard link remains unchanged after the name change because it points to the i-node, which maintains the data even if the file name is changed. See figure 3.3.8 and figure 3.3.9

```
vboxuser@HR:~/Desktop$ cat f2-soft
cat: f2-soft: No such file or directory
```

Figure 3.3.8 Display the contents of the soft link file.

```
vboxuser@HR:~/Desktop$ cat f1-hard
This is the first file
```

Figure 3.3.9 Display the contents of the hard link file.

A shell script that checks if the symbolic link exists, and if it does, prints the absolute path of the file it points to; otherwise, it displays an error message. See figure 3.3.10

```
#!/bin/bash

# Check if the symbolic link path is provided as an argument
if [ $# -ne 1 ]; then
    echo "Usage: $0 <absolute_path_to_symlink>"
    exit 1
fi

SYMLINK=$1

# Check if the symbolic link exists and is a valid symlink
if [ ! -L "$SYMLINK" ]; then
    echo "Error: The given path is either not a symbolic link or does not exist."
    exit 2
fi

# Get the absolute path of the target file
TARGET_FILE=$(readlink -f "$SYMLINK")

# Print the target file's absolute path
echo "The symbolic link '$SYMLINK' points to: $TARGET_FILE"
```

Figure 3.3.10 Bash script to check and resolve a symbolic link.

Display the result of a bash script, by execute the script in the terminal, see figure 3.3.11

```
vboxuser@HR:~/Desktop$ bash sym.sh
Usage: sym.sh <absolute_path_to_symlink>
vboxuser@HR:~/Desktop$
vboxuser@HR:~/Desktop$ bash sym.sh f1-hard
Error: The given path is either not a symbolic link or does not exist.
vboxuser@HR:~/Desktop$
vboxuser@HR:~/Desktop$
vboxuser@HR:~/Desktop$ bash sym.sh f2-soft
The symbolic link 'f2-soft' points to: /home/vboxuser/Desktop/f2
vboxuser@HR:~/Desktop$
```

Figure 3.3.11 Run the script named sym.sh once with no input, once when the file is not a soft link, and once when there is an input and it is a soft link.

3.4 File Permissions

In Linux, file permissions are controlled using three permission bits for three user categories:

- **User (u)**: The file owner.
- **Group (g)**: Users in the same group as the owner.
- **Others (o)**: which includes all other users.

Each category has three types of permissions:

- **Read**: Allows reading the file.
- **Write**: Allows modifying the file.
- **Execute**: Allows running the file.

The output of ls -l shows detailed information about the file, including its permissions, number of links, owner, group, size, last modification time, and file name. See figure 3.4.1

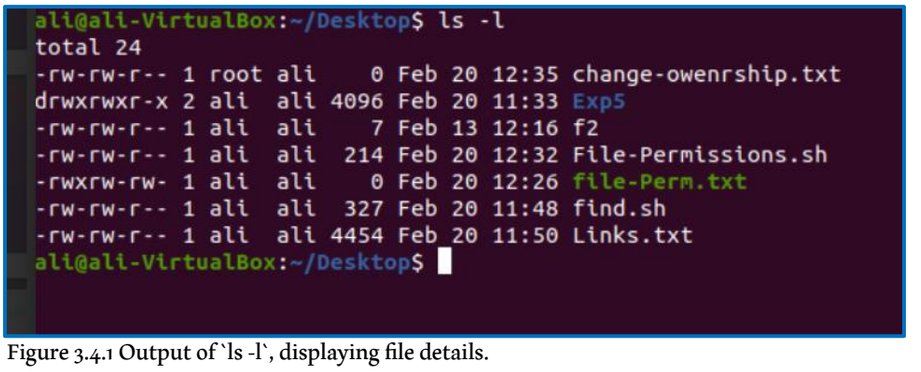

Figure 3.4.1 Output of `ls -l`, displaying file details.

Creating a file and set its permissions to allow read and write access for all users while making it executable only for the owner. See figure 3.4.2

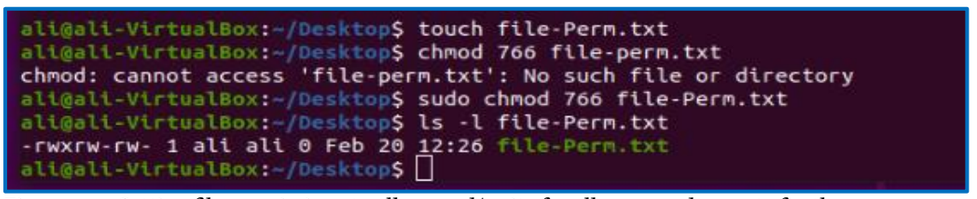

Figure 3.4.2 Setting file permissions to allow read/write for all users and execute for the owner.

The types of executable files are:

- **Binary Executables**: Files that contain instructions specific to the processor, which can be run directly.
- **Shell Scripts**: Text files containing a series of commands executed by the shell interpreter.
- **Python/Perl Scripts**: Text files containing code written in Python or Perl, executed by their respective interpreters.
- **Shared Object Libraries**: Files containing code that can be used by multiple programs.

Creating a Bash script that accepts the absolute path of a directory as an argument and lists all executable files owned by the user in that directory. Look at Figure 3.4.3 to see the Bash script, and refer to Figure 3.4.4 to observe how the script is executed and how it returns the expected results.

```
#!/bin/bash

if [ -z "$1" ]; then
    echo "22222222222"
    exit 1
fi

if [ ! -d "$1" ]; then
    echo "the desktop doesn't exists"
    exit 1
fi

echo "The executable files in  $1:"
find "$1" -type f -executable
```

Figure 3.4.3 Bash script to list executable files in a directory.

```
ali@ali-VirtualBox:~/Desktop$ touch File-Permissions.sh
ali@ali-VirtualBox:~/Desktop$ gedit File-Permissions.sh
ali@ali-VirtualBox:~/Desktop$ bash File-Permissions.sh /Desktop
the desktop doesn't exists
ali@ali-VirtualBox:~/Desktop$ bash File-Permissions.sh /home/ali/Desktop
The executable files in  /home/ali/Desktop:
/home/ali/Desktop/file-Perm.txt
ali@ali-VirtualBox:~/Desktop$
```

Figure 3.4.4 Executing the script and displaying executable files.

To change the ownership of a file so that the root user becomes its owner, the chown command is used. Look at Figure 3.4.5 to see how the command is executed and how the file ownership changes successfully.

```
ali@ali-VirtualBox:~/Desktop$ sudo chown root change-ownership.txt
chown: cannot access 'change-ownership.txt': No such file or directory
ali@ali-VirtualBox:~/Desktop$ sudo chown root change-owenrship.txt
ali@ali-VirtualBox:~/Desktop$ ls -l change-owenrship.txt
-rw-rw-r-- 1 root ali 0 Feb 20 12:35 change-owenrship.txt
```

Figure 3.4.5 Changing file ownership to root using chown command.

Changing file ownership is necessary in several cases, such as:

1. **Transferring Files Between Users**: When files are moved from one user to another, it's important to change ownership so the new user has proper access and control over the files.
2. **Changing Ownership of Web Server Files**: When a web server is set up, files owned by the web server user need to be changed to ensure the server can read, write, or execute them as needed.
3. **Changing File Ownership in Shared Directories**: In shared directories, the ownership might need to be updated to allow different users or groups to access and modify the files properly.

3.5 File Naming

To create a file with spaces in its name, enclose the name in double quotes (" ") . Similarly, to create a file whose name starts with a dash (-), use -- before the filename to prevent it from being interpreted as an option. Look at Figure 3.5.1

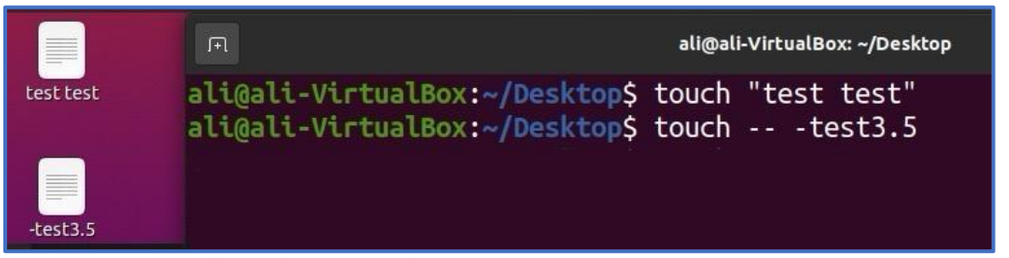

Figure 3.5.1 Creating files with spaces and leading dashes in their names.

To remove files with special characters or spaces in their names, use rm for files with spaces and files that starting with a dash (-), enclose the name in double quotes (" ") and use ./ before the filename to prevent it from being interpreted as an option. Look at Figure 3.5.2

Figure 3.5.2 Removing files with spaces and leading dashes in their names.

3.6 Sticky Bit

The sticky bit is a special permission (represented by t) primarily used on directories. It prevents users from deleting files within a directory unless they are the file's owner or the root user, even if other users have write permissions for the directory.

Display the permission details of the /tmp directory using the command ls -ld /tmp. Note that the last character (t) in drwxrwxrwt indicates that the Sticky Bit is set. This setting allows only the file owner to delete their own files, preventing other users from deleting each other's temporary files, Look at Figure 3.6.1 .

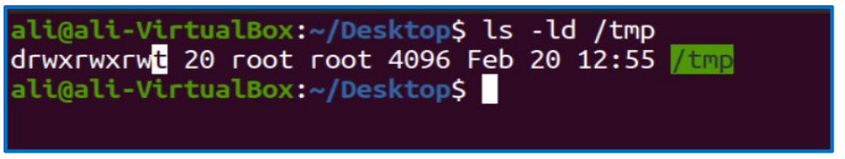

Figure 3.6.1 View the permissions of the /tmp, , showing the Sticky Bit (t) enabled.

Creating a Bash script that enables the Sticky Bit on any directory passed as an argument during execution, look at figure 3.6.2 .

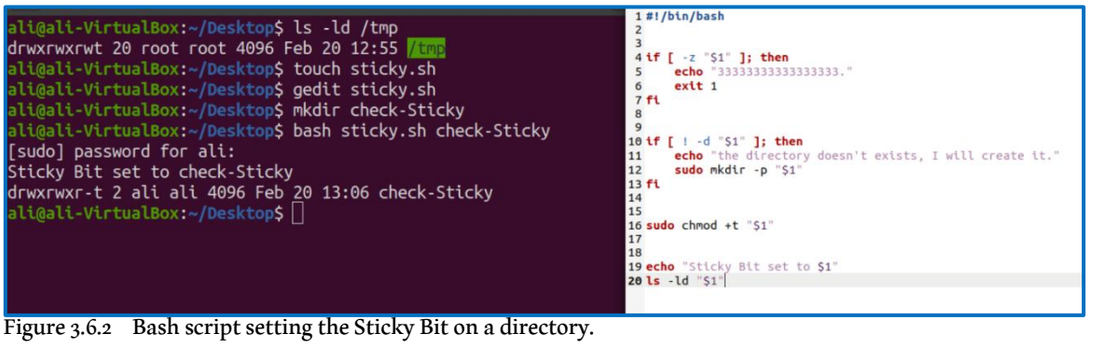

Figure 3.6.2 Bash script setting the Sticky Bit on a directory.

Challenges and Difficulties

The challenge we encountered was that the environment we were working on, which involved using Linux through Ubuntu VirtualBox, did not allow us to perform the mount command for Windows files as requested in requirement 3.1. To solve this issue, we added another virtual machine and mounted its files to our account. The details will be provided in the images attached below.

First, we shut down the Virtual Machine using ACPI Shutdown to allow us the opportunity to add the hardisk. See figure 3.7.1

Figure 3.7.1 The Virtual Machine is now powered off and ready for hard disk addition

Next, we went to the Virtual Machine settings, specifically to the Storage section, then to the Devices area. See figure 3.7.2

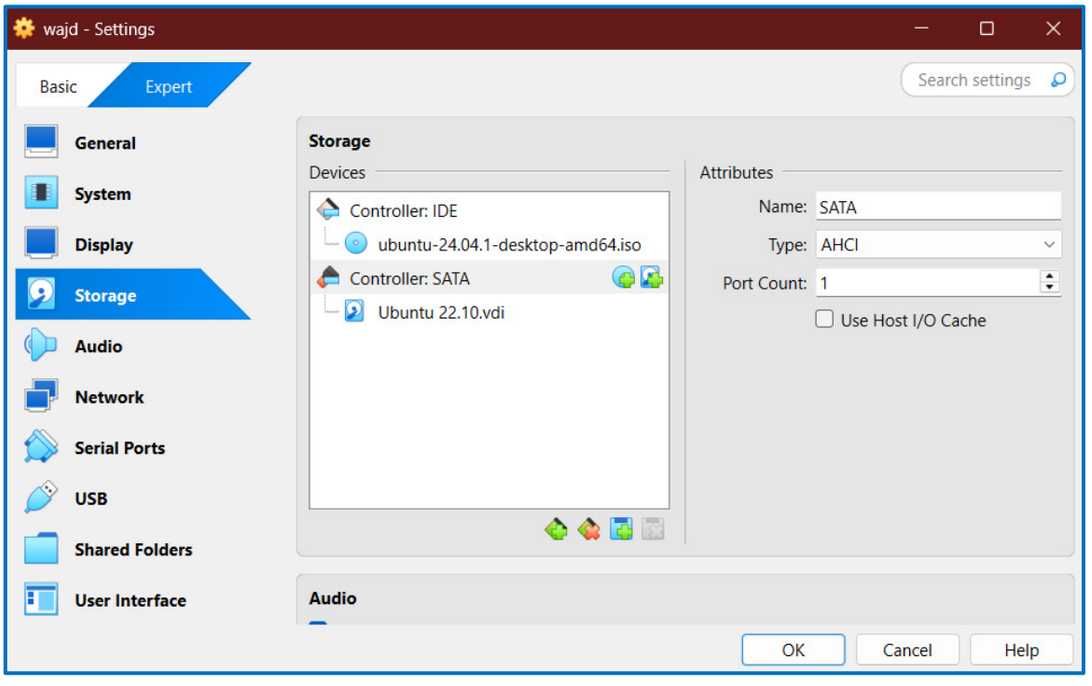

Figure 3.7.2 Storage area in the settings

Then, we clicked on the small disk icon to display a list from which we could choose the hard disk to add. See figure 3.7.3

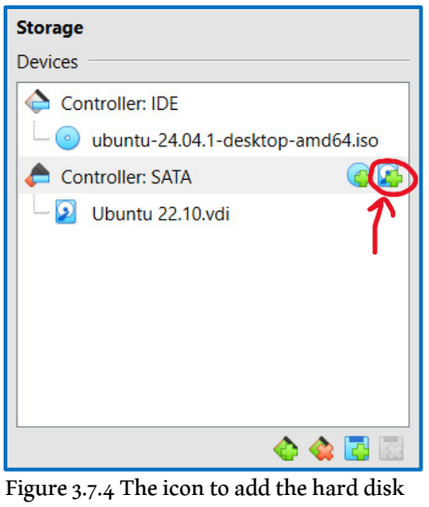

Figure 3.7.4 The icon to add the hard disk

Then, the list is displayed, and you can choose any option you want from the available choices. To confirm, either double-click on the desired option or click on the Choose icon. See figure 3.7.4

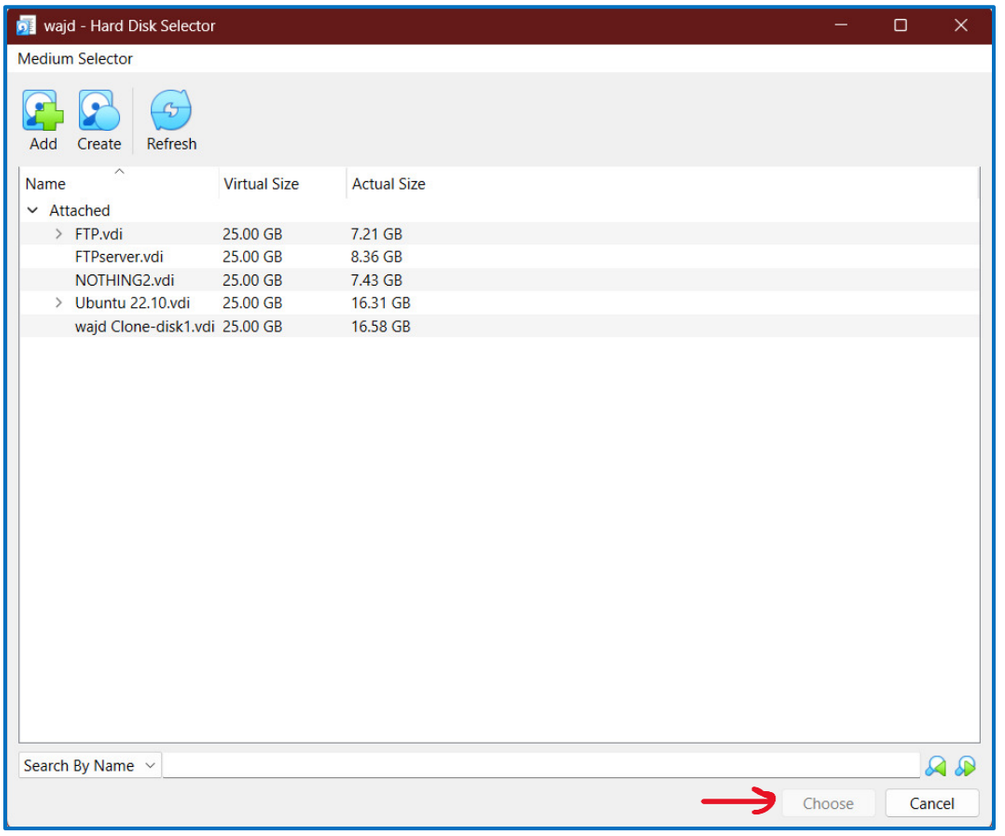

Figure 3.7.4 Hard Disk Selector

Conclusions

The experiment provided valuable hands-on experience in managing file systems and file permissions in Linux environments. Key takeaways include:

1. **Understanding File Systems:** We gained a deeper understanding of different file system types (e.g., NTFS, ext4) and how to manage them using tools like fdisk, mount, and fstab.
2. **File Permissions and Ownership:** We learned how to control file access using permissions (read, write, execute) and how to change file ownership, ensuring proper security and access control.
3. **Symbolic and Hard Links:** The experiment helped us differentiate between symbolic and hard links, demonstrating how they work and their use cases in file management.
4. **Sticky Bit Concept:** The sticky bit concept was useful in understanding how to protect files in shared directories, allowing only the owner or root to delete their files.
5. **Challenges and Problem Solving:** The experiment highlighted challenges, particularly with mounting Windows file systems on Linux. We successfully overcame these issues by adding a virtual machine and adjusting configurations.

Overall, the experiment enhanced our practical skills in network and system administration, reinforcing important concepts for managing secure and efficient file systems.

Done...