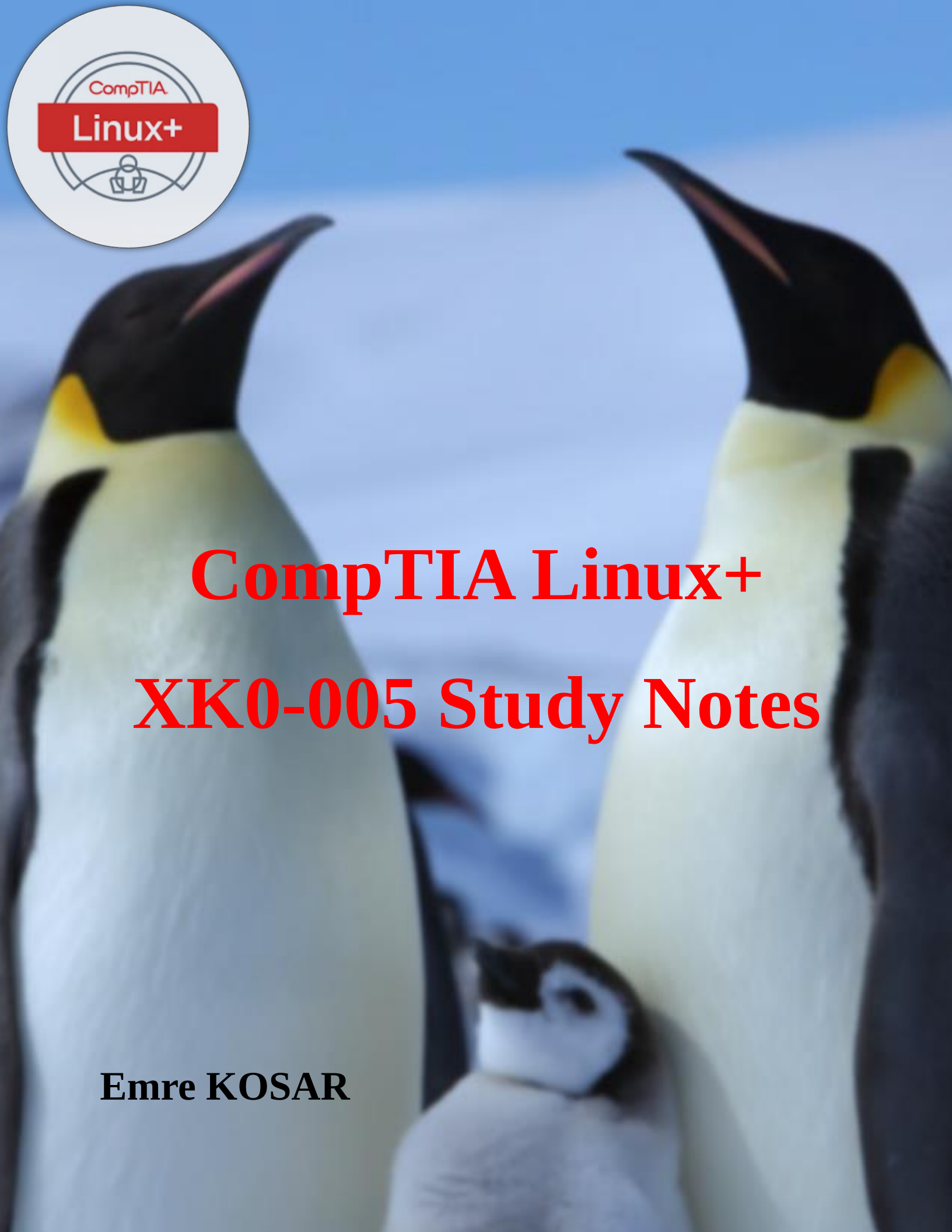


CompTIA Linux+

XK0-005 Study Notes

Emre KOSAR

Table of Contents

About CompTIA Linux+ Exam

In general, CompTIA Linux+ certification proves that we can securely administrate and keep the systems running in enterprise. To learn more about the exam, gently visit https://partners.comptia.org/docs/default-source/resources/comptia-linux-xk0-005-exam-objectives-(1-0).

The exam consists of a maximum of **90 questions** and **90 minutes**. Questions are typically multiple-choice questions and performance based. The passing score is **720** points on a scale of 100 to 900.

The exam includes 4 main domains. They are **System Management**, **Security**, **Scripting, Containers and Automation** and **Troubleshooting**.

DOMAIN	PERCENTAGE OF EXAMINATION
1.0 System Management	32%
2.0 Security	21%
3.0 Scripting, Containers, and Automation	19%
4.0 Troubleshooting	28%
Total	**100%**

System Management

- Summarize Linux Fundamentals.
- Given a scenario, manage files and directories.
- Given a scenario, configure and manage storage using the appropriate tools.
- Given a scenario, configure and use the appropriate process and services.
- Given a scenario, use the appropriate networking tools or configuration files.
- Given a scenario, build and install software.
- Given a scenario, manage software configurations.

Security

- Summarize the purpose and use of security best practices in a Linux environment.
- Given a scenario, implement identity management.
- Given a scenario, configure and execute remote connectivity for system management.
- Given a scenario, apply the appropriate access controls.

Scripting, Containers and Automation

- Given a scenario, create simple shell scripts to automate common tasks.
- Given a scenario, perform basic container operations.
- Given a scenario, perform basic version control using Git.
- Summarize common infrastructure as code (IaC) technologies.
- Summarize container, cloud, and orchestration concepts.

Troubleshooting

- Given a scenario, analyze and troubleshoot storage issues.
- Given a scenario, analyze and troubleshoot network resource issues.
- Given a scenario, analyze and troubleshoot Central Processing Unit (CPU) and memory issues.
- Given a scenario analyze and troubleshoot user access and file permissions.
- Given a scenario, use **systemd** to diagnose and resolve common problems with a Linux system.

Philosophy Behind the Design of Linux

-Linux is an open-source operating system, which means that it's totally free to use and it can be modified and redistributed. Linux is a Unix-like operating system which is typically packaged into distributions. Some of the popular distributions are Ubuntu, Fedora, Kali Linux, Parrot Linux, Linux Mint, Arch Linux, Debian, Red Hat etc.

- **Open-Source Software:** Source Code of open-source software can be viewed, modified and used by the public. **Linux** is an open-source software.
- **Proprietary Software:** It's the opposite of open-source software. It is licensed software that has restrictions on what end users can do. For instance, **macOS** is a proprietary software.

-Linux follows the Unix philosophy of simplicity and modularity. It makes Linux highly customizable, reliable and stable. Linux distributions are based on the Linux kernel.

Introduction to CLI and Shell

- **CLI (Command Line Interface):** Command Line Interface is a command line program which accepts text input to execute operating system functions.
- **Shell:** User interface used to interact with the operating system which allows the user to pass commands to the kernel. Some of the popular shell scripting languages are Bash (Bourne-Again Shell), zsh (Z-Shell), PowerShell (msh) etc.

-In Linux-based operating systems, shell scripting languages such as Bash, Korn Shell (ksh) etc. are **case sensitive**.

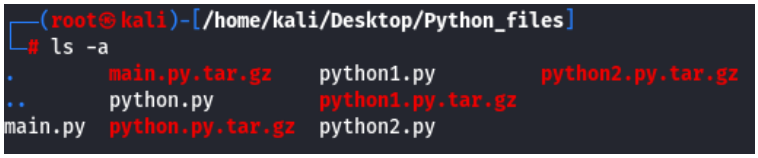
Introduction to Bash Commands

- **echo:** This command repeats the provided input back to the user as output.

- **ls:** This command lists what's inside of a directory. It can also show the file permissions and hidden files depending on the option.

-Here is how the output looks without any parameters:

-We can use **-a** parameter to show hidden files.



-We can both list and view hidden files using **-la** parameter (**-l** for *listing files with permissions*, **-a** for *hidden files*).

```
┌──(root㉿kali)-[/home/kali/Desktop/Python_files]
└─# ls -la
total 24
drwxr-xr-x  2 kali kali 4096 May  3 10:06 .
drwxr-xr-x 33 kali kali 4096 Aug  2 13:28 ..
-rw-r--r--  1 kali kali    0 May  3 10:03 main.py
-rw-r--r--  1 kali kali  145 May  3 10:06 main.py.tar.gz
-rw-r--r--  1 kali kali    0 May  3 10:04 python.py
-rw-r--r--  1 kali kali  146 May  3 10:06 python.py.tar.gz
-rw-r--r--  1 kali kali    0 May  3 10:04 python1.py
-rw-r--r--  1 kali kali  147 May  3 10:06 python1.py.tar.gz
-rw-r--r--  1 kali kali    0 May  3 10:04 python2.py
-rw-r--r--  1 kali kali  147 May  3 10:06 python2.py.tar.gz
```

- **pwd:** This command prints the current working directory. It stands for **print working directory**.

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# pwd
/home/kali/Desktop
```

- **cd:** This command allows us to change the current directory to provided directory. It stands for **change directory**.

-Changing directory to */home/kali/Downloads* directory.

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# cd /home/kali/Downloads/
```

-We can change back to *parent directory* by using **two dots** (**..**).

```
┌──(root㉿kali)-[/home/kali/Downloads]
└─# cd ..

┌──(root㉿kali)-[/home/kali]
└─# pwd
/home/kali
```

- **cp:** This command is used for copying files or directories to another location.

-Copying *copy_me.txt* file into the *Python_files* directory.

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# cp copy_me.txt Python_files/
```

- **mkdir:** This command allows us to create a *new directory*.

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# mkdir comptia_linux+
```

- **clear:** This command clears the terminal.
- **cat:** This command shows the *contents of a file* on the terminal page for reading.

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# cat linux+_cat.txt
cat command
showing you what's inside of it
that's it
best of luck
```

- **less:** This command is used to view the contents of a file that *won't fit on one the terminal screen*.

```
cat command
showing you what's inside of it
that's it
best of luck
linux+_cat.txt (END)
```

- **touch:** This command is used to create a *new file*.
- **vi/vim:** It's a *default text editor in Linux*. It can be written in both *vi* and *vim*.
- **nano:** It's a more simple and *user-friendly text editor* used in Linux.
- **gedit:** It's a *GUI based text editor* in Linux.
- **shutdown:** This command allows us to shut down the system.

-We can shut down the system immediately by using **-h** option with **now** argument.

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# shutdown -h now
```

- **reboot:** This command allows us to *reboot the system*. The *reboot* or *shutdown -r now* commands can be used to reboot.
- **su:** This command allows us to *switch user credentials*. Stands for *substitute user*. It will ask for password for authorization purposes.
- **history:** This command lists the *most recently entered command* as the output.
- **grep:** This command is used for *performing text searches* for a defined criteria of words or strings.

```
┌──(root㉿kali)-[/home/kali/Desktop/Python_files]
└─# ls -al | grep copy
-rw-r--r--  1 root root    0 Aug  6 16:26 copy_me.txt
```

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# grep -i it linux+_cat.txt
showing you what's inside of it
that's it
```

- **man:** This command is also known as *manpage*. It contains *documentations for Linux commands and tools*. So, it helps us to learn more about a command or tool in Linux.
- **apropos:** This command is used for *searching manpage database*.
- **whatis:** This command is used to display a *brief description of the given commands*.
- **info:** This command is used to display the *information page of a command*.

-Additionally, we can use **-h** or **--help** to see how that specific command works.

Users and Groups in Linux

-Essentially, users and groups are used for security/access control purposes in Linux. Users and Groups help us to obtain who can access specific files, directories etc. There are three diverse types of user accounts.

1. ***Root Account:*** Can do **administrative tasks**. It should *not be shared* with anyone. Some of the administrative tasks that can be done by root account are *password reset, installing and updating software packages, modifying configuration settings on the system, managing user accounts* etc.
2. ***Standard Account:*** Also known as **regular user account**, interacts with the system. Unlike the root account, standard accounts have *limited privileges* and cannot perform tasks that require administrative privileges.
3. ***Service Account:*** Also known as **system account**, created on a Linux system to run a specific service or application. Such as *HTTP web service (Apache Web Service), mySQL for database service*. Unlike

standard accounts, service accounts are used specifically for services and have no interactive login shell.

-In Linux systems, users with administrative credentials are known as the **superuser**. It's also named **root**.

- **su:** This command allows to switch the provided user credentials. To switch to the root user, we can use **su -root** command.
- **sudo:** This command enables us to execute a command as *superuser (root)* after authenticating the current user. The user has to be delegated in the */etc/sudoers*.
- **sudoedit:** This command *permits a user to edit a file with their own credentials* even if the file is only available to the root user.

-Keep in mind that we *should NOT* edit **/etc/sudoers** file with standard text editors like nano, vim or gedit. Only use **visudo** (*visudo /etc/sudoers*). The visudo command verifies /etc/sudoers syntax before committing changes.

- **visudo:** This command is used to edit the *sudoers* file. It has a few options. The **-c** option checks file errors, **-f** option edits/checks location, **-s** option checks file in strict mode, and **-x** option shows output in JSON format.

-Many distributions will disable the actual root account for users and instead it allows administrative functions based on membership in what's known as the **wheel group**.

- **wheel group:** Members exercise root privileges with less potential for damaging the system. For instance, members of the wheel group can use the *sudo* command to avoid having to sign in as the root user.

-We can also use the *visudo* command to edit the privileges of the wheel group.

- **Polkit (PolicyKit):** Controls system-wide privileges that allows non-privileged processes to communicate with privileged ones. Polkit allows a command to execute with privileges using the command **pkexec** followed by the command we want to execute.

Creating, Modifying and Deleting Users

- **useradd:** This command is used to *create a new user account*. It has different options. Such as the **-m** option to create the user's home directory and the **-s** option to specify the default shell. Additionally, this command does not set a password for the account.
- **adduser:** This command is an *interactive way to create a new user account*. It's also more user-friendly.
- **passwd:** This command is used by the root user to *set* or *reset a password* for any user in the system. A user can change their own password by using this command.
- **chage:** This command is used to change the attributes for user account. Such as *expiration date*, *password expiration information*. The chage command *requires root permission*.

- All of the user passwords are stored in the file called ***/etc/shadow***. This file contains *hashed passwords* and additional account information. Only the **root** user has access to the content of /etc/shadow.

-The **/etc/shadow** file contains several fields, these are: *Username*, *Password in hashed format*, *Days since password was changed*, *Days before password must be changed*, *Days until user is warned to change password*, *Days after password expires that account gets disabled*, *Days the account has been disabled*, and *Unused field that is reserved for future use*.

-Inside of the **/etc/login.defs** file, we can see some configuration settings for the shadow password suite, which is used to manage user accounts and passwords on Linux.

-Keep in mind that regular users will get an ID *between 1,000 and 60,000*. Root users will get an ID as **zero**.

-The **sudo !!** command allows us to call the last command.

Hands-on Experience – useradd, chage, usermod, passwd, userdel

-We can add user by using **useradd** command.

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# useradd -u 1453 -c "comptia linux+" -d /home/linux -s /bin/bash -p 12345678 -e 08/11/2024 linux
```

- In this case, new user named **linux** is being created on the system with some options. The **-u** option specified its UID (User ID) as *1453*, **-c** option added comment as *comptia linux+*, **-d** option specified a home directory as */home/linux* for the user, **-p** option set the encrypted password as *12345678* (not recommended), and **-e** parameter set an expiry date as *11th of August 2024*.

-We can check if the user is created or not by reading **/etc/passwd** or **/etc/shadow** file.

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# tail -n 2 /etc/passwd
ekosar:x:1002:1002:Emre Kosar:/home/ekosar:/bin/sh
linux:x:1453:1453:comptia linux+:/home/linux:/bin/bash
```

- In this case, on the last line we can see some information about the user (linux) created previously, such as username, UID and GID (separeted by colon), comment, home directory and shell by this order. Each of the information is separated by colon.

-We can change the expiration date by using **chage** command with **-E** parameter.

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# chage -E 08/15/2024 linux
```

-To see the expiration date and some other aging information of an account, we will be using **chage** command with **-l** parameter.

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# chage -l linux
Last password change                                    : Aug 09, 2024
Password expires                                        : never
Password inactive                                       : never
Account expires                                         : Aug 15, 2024
Minimum number of days between password change          : 0
Maximum number of days between password change          : 99999
Number of days of warning before password expires       : 7
```

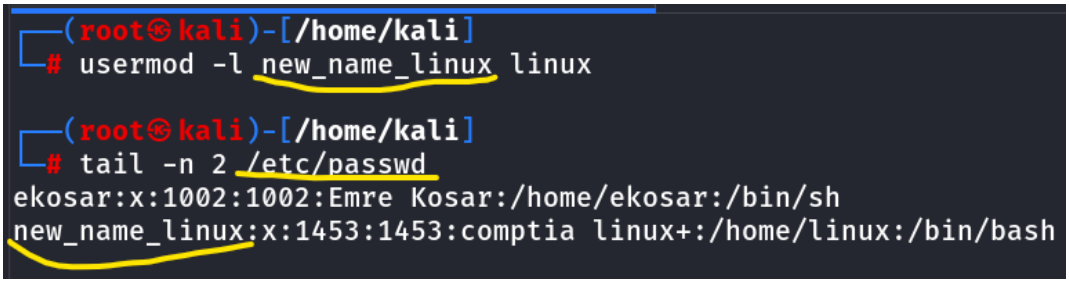
-We can set the maximum days to change a user's password by using **chage** command with **-M** option.

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# chage -M 30 linux

┌──(root㉿kali)-[/home/kali/Desktop]
└─# chage -l linux
Last password change                                    : Aug 09, 2024
Password expires                                        : Sep 08, 2024
Password inactive                                       : never
Account expires                                         : Aug 15, 2024
Minimum number of days between password change          : 0
Maximum number of days between password change          : 30
Number of days of warning before password expires       : 7
```

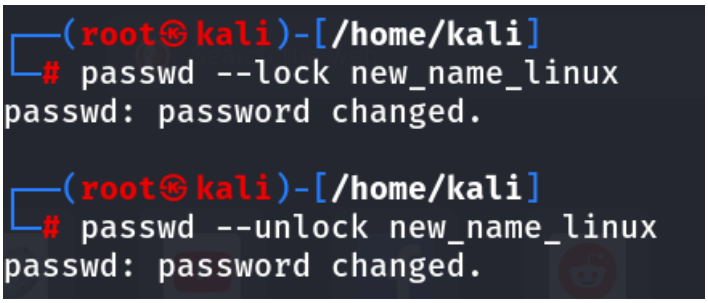
-We can set a warning to the user before password expiration by using **chage** command with **-W** option.

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# chage -W 3 linux

┌──(root㉿kali)-[/home/kali/Desktop]
└─# chage -l linux
Last password change                                    : Aug 09, 2024
Password expires                                        : Sep 08, 2024
Password inactive                                       : never
Account expires                                         : Aug 15, 2024
Minimum number of days between password change          : 0
Maximum number of days between password change          : 30
Number of days of warning before password expires       : 3
```

-To change the name of the user, we need to use **usermod** command with **-l** parameter.

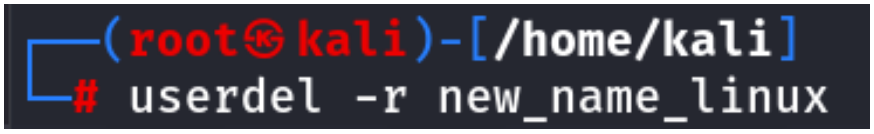


-We can lock a user by using **passwd** command with **-l** or **--lock** parameter. Additionally, we can unlock it by using **-u** or **--unlock** parameter.



-To delete a user, we can use **userdel** command.

-Even if the user is deleted, we can still be able to see their directory listed under /home/. In order to delete everything about a user, we need to use **userdel** command with **-r** option.



Creating, Modifying and Deleting Groups

-Groups in Linux simply perform administrative tasks, and they're represented on the system by a *group ID number* (**GID**). Users in Linux can be members of more than one group.

-To find information related to groups on the system, we will be referencing **/etc/group** file because it's the storage location for all of the groups. Each group contains four fields of information separated by colons:

1. ***Group Name*:** Refers to user-friendly name of the group.
2. ***Password*:** Refers to password required to enter a group (Usually 'x').
3. ***Group ID (GID)*:** Refers to reference number on the system.
4. ***Group List***: Refers to members of the group. By default, group list will be empty.

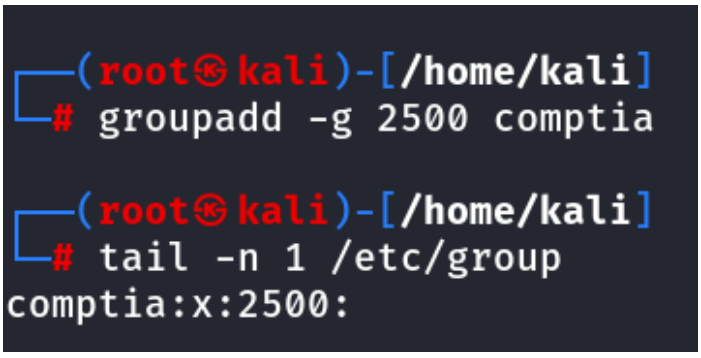
- **groupadd:** This command is used to *create a group*. By default, the group has *no members* and *no password*. It has several options. Such as, **-g** option creates a group with specified GID, **-o** option allows us to create a group with non-unique GID.

-Manually editing */etc/group* file is not recommended. Instead, we should use **groupmod** command.

- **groupmod:** This command is used to *change a group's attributes*. It has a few options. Such as, **-g** option helps us to change the group ID (GID), **-n** option allows us to rename the group.
- **groupdel:** This command is used to *delete groups*. It won't be deleting user accounts within a group, only deleting the group itself.

Hands-on Experience – groupadd, groupmod, groupdel

-We can create group by using **groupadd** command.



- In this case, a new group named *comptia* is being created on the system with the GID value as *2500*. Additionally, we can check if the group is being created or not by viewing **/etc/group** file.

-We can change the group name by using **groupmod** command with **-n** parameter.

```
┌──(root㉿kali)-[/home/kali]
└─# groupmod -n CompTIA comptia
```

- In this case, the group name has been changed from comptia to *COMPTIA*.

-We can add a user to a group by using **usermod** command with **-aG** parameter. The **-aG** parameter stands for *Append Group*.

```
┌──(root㉿kali)-[/home/kali]
└─# usermod -aG CompTIA linux+ ; usermod -aG CompTIA security+ ; usermod -aG CompTIA pentest+ ; usermod -aG CompTIA network+

┌──(root㉿kali)-[/home/kali]
└─# tail -n 1 /etc/group
CompTIA:x:2500:linux+,security+,pentest+,network+
```

- In this case, *linux+*, *security+*, *pentest+*, and *network+* users have been added into **COMPTIA** group.

-We can delete a group by using **groupdel** command. The critical point is that when we delete a group, we won't be deleting the members of the group.

```
┌──(root㉿kali)-[/home/kali]
└─# groupdel CompTIA

┌──(root㉿kali)-[/home/kali]
└─# tail -n 1 /etc/group
network+:x:1005:

┌──(root㉿kali)-[/home/kali]
└─# tail -n 5 /etc/passwd
systemd-resolve:x:987:987:systemd Resolver:/:/usr/sbin/nologin
security+:x:1002:1002:Cyber Security:/home/security+:/bin/sh
pentest+:x:1003:1003:Pen Test:/home/pentest+:/bin/sh
linux+:x:1004:1004:Sys Admin:/home/linux+:/bin/sh
network+:x:1005:1005:Network Specialist:/home/network+:/bin/sh
```

Gathering Information About Users and Groups

- **whoami:** This command is used to display the *user currently logged in the system*. We can use this command to verify the current username.

-When we are logged in as the **root user**, we will see **#** sign on our command prompt. When we are logged in as **standard user**, we will see **$** sign on our command prompt.

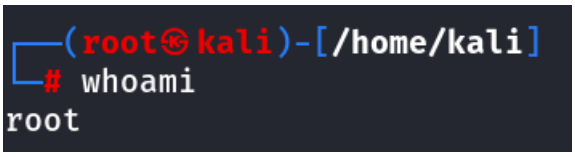
```
┌──(kali㉿kali)-
└─$
```

```
┌──(root㉿kali)-[/home/kali]
└─#
```

- **who:** This command is used to determine the details of the users *currently logged* in the system. The output of this command consists of the *username*, *name of the system*, and the *date and time of the user's been connected since*. This command has several options. Such as, **-b** parameter displays time of the last system boot, **-u** parameter shows how long users have been connected but haven't done anything. If we see *a dot* (.), this indicates the user was acting up until the last minute. The **old** section indicates that the users have been *inactive for over 24 hours*.
- **w:** This command is used to display the *details of users that are currently logged in* to a system and their transactions. It has several options. The **first line of the output** displays the *status of the system*. The **second line of the output** displays *a table column list of the users logged in to the system*. The **last column** indicates the *current activities of the users*.
- **last:** This command displays the *history of user login and logout actions*, and the *actual time and date*. This command retrieves information from the **/var/log/wtmp** file.
- **id:** This command is used to display *user ID* (UID) and *group ID* (GID) information. Using this command without any options displays information about the user who is currently logged in.

Hands-on Experience – whoami, id, who, w, last

-We can display the current user by using **whoami** command.



-We can display the ID information of the current user by using **id** command.

```
┌──(root㉿kali)-[/home/kali]
└─# id
uid=0(root) gid=0(root) groups=0(root)
```

- If we see the UID and GID of the current user equals zero (0), this indicates that the user is **ROOT**. So, keep in mind that if **UID** and **GID** equals zero, it always indicates **ROOT**.

-We can display the ID information of another user by using **id** command with username.

```
┌──(kali㉿kali)-[~]
└─$ id comptia
uid=1002(comptia) gid=1002(comptia) groups=1002(comptia)
```

-We can display the users who currently logged in to the system by using **who** command.

```
┌──(root㉿kali)-[/home/kali]
└─# who
kali     seat0        2024-08-12 10:23 (login screen)
kali     :1           2024-08-12 10:23 (:1)
kali     pts/1        2024-08-12 10:30
kali     pts/4        2024-08-13 03:51
```

-We can display detailed information about currently logged in users into the system by using **w** command.

```
┌──(root㉿kali)-[/home/kali]
└─# w
 09:56:58 up  7:38,  1 user,  load average: 0.28, 0.26, 0.26
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
kali     tty2     -                Mon10   23:40m  1:05   0.05s /usr/libexec/gnome-session-binary
```

-We can display the history of the user login and logout actions by using **last** command.

```
┌──(root㉿kali)-[/home/kali]
└─# last
kali     :1           :1               Mon Aug 12 10:23    gone - no logout
kali     seat0        login screen     Mon Aug 12 10:23    gone - no logout
reboot   system boot  6.5.0-kali3-amd6 Mon Aug 12 10:16   still running
kali     :1           :1               Mon Aug 12 08:32 - down   (00:48)
kali     seat0        login screen     Mon Aug 12 08:32 - down   (00:48)
reboot   system boot  6.5.0-kali3-amd6 Mon Aug 12 08:29 - 09:21  (00:52)
kali     :1           :1               Tue Aug  6 16:01 - down  (4+20:45)
kali     seat0        login screen     Tue Aug  6 16:01 - down  (4+20:45)
reboot   system boot  6.5.0-kali3-amd6 Tue Aug  6 15:26 - 12:47 (4+21:20)
kali     :1           :1               Mon Aug  5 15:08 - down   (00:04)
```

Configuring Account Profiles

-Administrators have tasks like what software users can install, what files or directories users can read/write/execute etc.

- ***.bashrc file:*** This file enables customization of the *user's own environment*. For example, users can specify *their own abbreviated commands* without impacting anybody else, this is called **alias**. Users can *create environment variables*. Users can *set default directories and file permissions*. Users can *change default command prompt*.
- ***.bash_profile file:*** This file provides the *shell configuration for the initial login environment*. So, when we change the .bash_profile file, it will be reading with the first login. After that, whatever we change in the .bash_profile file, think about that as part of the *skeleton file* (**/etc/skel/** directory) when we create a new account.

- When a user is being created on the system, the system **automatically copies** all of the contents inside **/etc/skel/** directory to new user's home directory.

-Any files added to /etc/skel/ directory after a user account is created will **not be copied** to existing user's home directory.

-When we also want to apply certain settings to accounts for all the users on a system, we should use **/etc/profile** file.

- **/etc/profile file:** This file provides system-wide environment variables that are used to *apply certain settings to user accounts*.

- **/etc/profile.d/ directory:** This directory serves as a *storage location for scripts* that admins may use to set additional system-wide variables. It is recommended that we set the environmental variables via scripts contained in the **/etc/profile.d/** directory rather than /etc/profile file.
- **/etc/bash.bashrc file:** This file provides system-wide configuration changes *specific to Bash settings* (command line environment).

Hands-on Experience – .bashrc, .bash_profile, /etc/skel

-We can customize our Bash Shell environment inside of the **.bashrc** file.

1. First, open the .bashrc file with a text editor (such as vim, nano etc.).
2. Find "Alias Definitions" or "User Specific Aliases" or similar things like that.
3. Simply add your alias(es) and save it.
4. Apply the changes in .bashrc file by using **source .bashrc** command.

```
# Alias definitions.
alias vpn="cd /home/kali/Desktop/vpnbook-openvpn-pl134 ; openvpn vpnbook-pl134-tcp443.ovpn"
alias emre="ls -la /etc | grep cron"
```

```
┌──(root㉿kali)-[/home/kali]
└─# source .bashrc
┌──(root㉿kali)-[/home/kali]
└─# emre
drwxr-xr-x   2 root     root       4096 Feb 20 08:51 cron.d
drwxr-xr-x   2 root     root       4096 Jun 30 07:55 cron.daily
drwxr-xr-x   2 root     root       4096 Feb 20 08:49 cron.hourly
drwxr-xr-x   2 root     root       4096 Feb 20 08:49 cron.monthly
drwxr-xr-x   2 root     root       4096 Feb 20 08:51 cron.weekly
drwxr-xr-x   2 root     root       4096 Feb 20 08:49 cron.yearly
-rw-r--r--   1 root     root       1042 Nov 22  2023 crontab
```

-We can define environment variables, set up the user's shell environment etc. inside of the **.bash_profile** file.

1. First, open the .bash_profile file with a text editor (vim, nano etc.).
2. Add customization options as you wish, save and exit.
3. Apply changes by using **source ~/bash_profile** command.

```
 GNU nano 7.2                                                /root/.bash_profile
# ,bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

#User specific environment and startup programs

PATH=$PATH:$HOME/.local/bin:$HOME/bin

export PATH

export EDITOR=nano

source ~/.bash_rc
```

- In this case, the **PATH** variable is used to let the environment know where we are going to store all of our commands. So, Linux checks according to this PATH variable when a command is executed on the system.

-Let's say we have to specify policies for every user created on the system. Instead of doing it manually on every new user, we can just create a file in **/etc/skel/** directory; since it's automatically being copied to the new user's home directory.

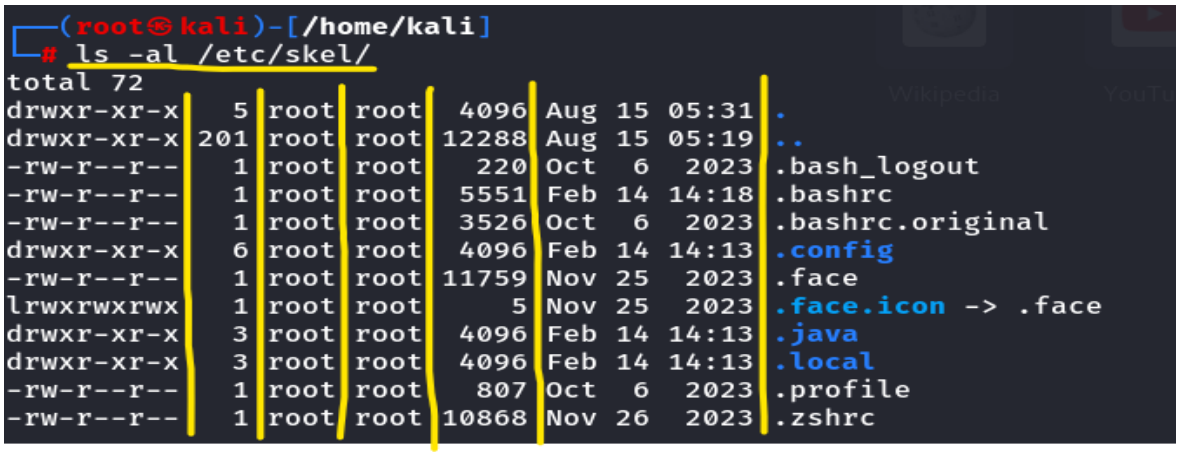
```
┌──(root㉿kali)-[/home/kali]
└─# touch /etc/skel/company_policy.txt
```

```
 GNU nano 7.2                                          /etc/skel/company_policy.txt
# This is the policy file for our company.

# Policies

alias sudo="/home/$USER/Desktop"
```

Permissions and Ownership in Linux

Permissions

-Permission is access rights assigned to users which enables them to access or modify files and directories. The **ls -l** command displays the permissions of a file or directory.



-The output of **ls -al** (-al option is a common way) command consists of seven fields. Let's examine it.

1. The **first column** shows if the item is a *file (-)* or *directory (d)*, *the user*, *group* and *other* permission assigned, and the access method.
2. The **second column** refers to the *number of links* there are.
3. The **third column** displays the *owner* of the *file* or *directory*.
4. The **fourth column** displays the *group* that has been granted access by the administrator.
5. The **fifth column** lists the *size* in *bites* of the *file* or *directory*.
6. The **sixth column** displays the *date and time* that the item was created or last modified.
7. The **seventh column** displays the *item's name*.

- Permission attributes explanations for files:

  -The **read** permission is known as **r** and it allows to *view the content*.
  -The **write** permission is known as **w** and it allows to *save changes*.
  -The **execute** permission is known as **x** and it allows to *run the file*.

- Permission attributes explanations for directories:

  -The **read** permission is known as **r** and it allows to *list the content*.
  -The **write** permission is known as **w** and it allows to *create, rename or delete files* in that directory.
  -The **execute** permission is known as **x** and it allows to access a directory and execute a file from that directory.

- Permission attributes on files and directories are applied to one of several contents, these are the **owner** known as **u**, the **group** known as **g** and **others** known as **o**.

  -The *owner* (***u***) refers to the *owner* of the file or directory.

  -The *group* (***g***) refers to the *file* or *directory's group*.

  -The *others* (***o***) refer to *other users* on the system. They are **not** owners or any users under the same group as the owner.

```
┌──(root㉿kali)-[/home/kali/Desktop/comptia_linux+_exam]
└─# ls -l
total 16
-rw-r-xrw- 1 root root   22 Aug 15 10:31 executable.py
-rw-rw-r-- 1 kali kali   57 Aug 15 10:27 linux.txt
drwxr-xr-x 2 kali kali 4096 Aug 15 10:27 permissions
-rw-r--r-- 1 root root   44 Aug 15 10:28 root_owner.txt
  u  g  o
```

-The **first character** circled with yellow refers to the *item type* (**-** for *file*, **d** for *directory*, **l** for *linked file*).

-The **second**, **third** and **fourth characters** (underlined with green) refer to **owner permissions** (In this case, it's **rw-**).

-The **fifth**, **sixth** and **seventh characters** (underlined with green) refer to **group permissions** (In this case, it's **r--**).

-The **eighth**, **ninth** and **tenth characters** (underlined with green) refer to **other permissions** (In this case, it's **r--**).

- **chmod:** This command allows the owner to *modify the permissions* of a file or directory. Only the *owner* or *root* can change the permissions of a file

or directory. It has several options. For example, **-c** option reports the changes, **-f** option hides error messages, **-v** option displays diagnostic entry for every file process, **-R** option modifies the permissions of files and directories recursively (which means it will go downward in the directory structure all the way to the bottom). The chmod command supports two different modes:

1. **Symbolic Mode:** It enables to set permissions using three components which includes permission context.
   -For permission context, we use **u (owner)**, **g (group)**, **o (other)**, **a (all)**.
   -For permission operators, we use **+ (add permissions)**, **- (deny permissions)**, **= (exactly assign permissions)**.
   -For permission attributes, we use **r (read)**, **w (write)**, **x (executable)**.
2. **Absolute Mode:** It uses octal (base-8) numbers to specify permissions. For **reading**, we use **4**. For **writing**, we use **2**. For executable, we use **1**. For **all** permissions, we use **7 (4+2+1=7)**

- **umask:** This command is used to set the default permissions for newly created files and directories.

Hands-on Experience – chmod, umask

-If we see an item written in **blue** on the output of **ls -l** command, it indicates that the item is a *directory*. If the color is **green**, it indicates that item is an *executable file*.

```
┌──(root㉿kali)-[/home/kali/Desktop/comptia_linux+_exam]
└─# ls -l
total 16
-rw-r-xrw- 1 root root   22 Aug 15 10:31 executable.py
-rw-rw-r-- 1 kali kali   57 Aug 15 10:27 linux.txt
drwxr-xr-x 2 kali kali 4096 Aug 15 10:27 permissions
-rw-r--r-- 1 root root   44 Aug 15 10:28 root_owner.txt
```

-Let's examine the permissions in detail.

```
┌──(root㉿kali)-[/home/kali/Desktop/comptia_linux+_exam]
└─# ls -l
total 16
-rw-r-xr-- 1 root root   22 Aug 15 10:31 executable.py
-rw-r----x 1 kali kali   57 Aug 15 10:27 linux.py
drwxrwxr-x 2 kali kali 4096 Aug 15 10:27 permissions
-rw-rw-rw- 1 kali kali   44 Aug 15 10:28 user.txt
```

- Consider the first item (*executable.py*). This item is owned by **root** and the permissions of this item are:

  -rw-r-xrw- → It's a *file*. The owner can **read** and **write** (rw-) and it's represented as **6** (4+2) in numerical. The group can **read** and **execute** (r-x) and it's represented as **5** (4+1) in numerical. Others can only **read** (r--) and it's represented as **4** in numerical. The numeric display of this file is **654**.

- Consider the second item (*linux.txt*). This item is owned by **user (kali)** and the permissions of this item are:

  -rw-r----x → It's a *file*. The owner can **read** and **write** (rw-) and it's represented as **6** (4+2) in numerical. The group can only **read** (r--) and it's represented as **4** in numerical. Others can only **execute** (--x) and it's represented as **1** in numerical. The numeric display of this file is **641**.

- Consider the third item (*permissions*). This item is owned by **user (kali)** and the permissions of this item are:

  drwxrwxr-x →It's a directory. By default, Linux implies **775** as permissions for directories. So, the owner and group can **read**, **write** and **execute (4+2+1=7)** where the others can **read** and **execute (4+1=5)**.

- Consider the fourth item (*user.txt*) This item is owned by **user (kali)** and the permissions of this item are:

-rw-rw-rw- →It's a file. By default, Linux implies **666** as permissions for files. So, all of the users (owner, group and others) can **read** and **write (4+2=6)**.

-We can change the permissions of a file or directory by using **chmod** command.

```
┌──(root㉿kali)-[/home/kali/Desktop/comptia_linux+_exam]
└─# ls -l
total 16
-rw-r-xr-- 1 root root   22 Aug 15 10:31 executable.py
-rw-r----x 1 kali kali   57 Aug 15 10:27 linux.py
drwxrwxr-x 2 kali kali 4096 Aug 15 10:27 permissions
-rw-rw-rw- 1 kali kali   44 Aug 15 10:28 user.txt

┌──(root㉿kali)-[/home/kali/Desktop/comptia_linux+_exam]
└─# chmod g+x linux.py

┌──(root㉿kali)-[/home/kali/Desktop/comptia_linux+_exam]
└─# chmod 651 linux.py
```

- In this case, both the **chmod g+x linux.py** and **chmod 651 linux.py** commands did the same thing. They granted executable permission for the group.

-Let's deny permission to an item in both ways.

```
┌──(root㉿kali)-[/home/kali/Desktop/comptia_linux+_exam]
└─# ls -l
total 16
-rw-r-xr-- 1 root root   22 Aug 15 10:31 executable.py
-rw-r-x--x 1 kali kali   57 Aug 15 10:27 linux.py
drwxrwxr-x 2 kali kali 4096 Aug 15 10:27 permissions
-rw-rw-rw- 1 kali kali   44 Aug 15 10:28 user.txt

┌──(root㉿kali)-[/home/kali/Desktop/comptia_linux+_exam]
└─# chmod o-x linux.py

┌──(root㉿kali)-[/home/kali/Desktop/comptia_linux+_exam]
└─# chmod 650 linux.py
```

- In this case, both the **chmod o-x linux.py** and **chmod 650 linux.py** commands did the same thing. They denied executable permission for others.

-Let's execute **umask** command.

```
┌──(root㉿kali)-[/home/kali/Desktop/comptia_linux+_exam]
└─# umask
0022
```

- In this case, we can notice the value as **0022**. The *first digit* is *zero* so there are **no permissions** that are set. The *second digit* sets permissions for **users who own the file**. A value of **zero** means **all permissions are allowed**. The *third digit* sets permissions for the **group that owns the file**. The value of **two** means **write permission is masked out**. The *fourth digit* sets permissions for **others**. The value of two means **write permission is masked out**.

-We can change the default **umask val**ue on **.bashrc** file.

```
#User specific
umask 0022
```

Ownership

-Ownership refers to the property by which a *user can apply* and *modify the permissions of a file* or *directory*. Only the root (super user) can change the permissions of an object owned by others.

- **chown:** This command is used to change the *owner* or *group* of a file or directory. We can *both* change the owner and group, or just change *one of them*. The **-R** option is used to change the ownership throughout a directory structure recursively.
- **chgrp:** This command is used to change the *group ownership* of a file or directory.

Hands-on Experience – chown, chgrp

-We can use the **chown** command with **-R** option to change ownerships of files in a specific directory.

```
┌──(root㉿kali)-[/home/kali/Desktop/comptia_linux+_exam]
└─# ls -l
total 16
-rw-r-xr-- 1 root root   22 Aug 15 10:31 executable.py
-rw-r-x--- 1 kali kali   57 Aug 15 10:27 linux.py
drwxrwxr-x 2 kali kali 4096 Aug 15 10:27 permissions
-rw-rw-rw- 1 kali kali   44 Aug 15 10:28 user.txt

┌──(root㉿kali)-[/home/kali/Desktop/comptia_linux+_exam]
└─# chown -R kali:kali . ; ls -l
total 16
-rw-r-xr-- 1 kali kali   22 Aug 15 10:31 executable.py
-rw-r-x--- 1 kali kali   57 Aug 15 10:27 linux.py
drwxrwxr-x 2 kali kali 4096 Aug 15 10:27 permissions
-rw-rw-rw- 1 kali kali   44 Aug 15 10:28 user.txt
```

-We can use **chgrp** command with **-R** option to change the group ownership of files in a specific directory.

```
┌──(root㉿kali)-[/home/kali/Desktop/comptia_linux+_exam]
└─# ls -l
total 16
-rw-r-xr-- 1 kali kali   22 Aug 15 10:31 executable.py
-rw-r-x--- 1 kali kali   57 Aug 15 10:27 linux.py
drwxrwxr-x 2 kali kali 4096 Aug 15 10:27 permissions
-rw-rw-rw- 1 kali kali   44 Aug 15 10:28 user.txt

┌──(root㉿kali)-[/home/kali/Desktop/comptia_linux+_exam]
└─# chgrp -R comptia . ; ls -l
total 16
-rw-r-xr-- 1 kali comptia   22 Aug 15 10:31 executable.py
-rw-r-x--- 1 kali comptia   57 Aug 15 10:27 linux.py
drwxrwxr-x 2 kali comptia 4096 Aug 15 10:27 permissions
-rw-rw-rw- 1 kali comptia   44 Aug 15 10:28 user.txt
```

-We can change a file's ownership or group ownership by using **chown** command.

```
┌──(root㉿kali)-[/home/kali/Desktop/comptia_linux+_exam]
└─# chown :kali user.txt

┌──(root㉿kali)-[/home/kali/Desktop/comptia_linux+_exam]
└─# ls -l
total 16
-rw-r-xr-- 1 kali comptia   22 Aug 15 10:31 executable.py
-rw-r-x--- 1 kali comptia   57 Aug 15 10:27 linux.py
drwxrwxr-x 2 kali comptia 4096 Aug 15 10:27 permissions
-rw-rw-rw- 1 kali kali      44 Aug 15 10:28 user.txt
```

- The **chown :kali user.txt** command only changed the group as *kali*, not the owner.

```
┌──(root㉿kali)-[/home/kali/Desktop/comptia_linux+_exam]
└─# chown comptia: linux.py

┌──(root㉿kali)-[/home/kali/Desktop/comptia_linux+_exam]
└─# ls -l
total 16
-rw-r-xr-- 1 kali     comptia   22 Aug 15 10:31 executable.py
-rw-r-x--- 1 comptia comptia    57 Aug 15 10:27 linux.py
drwxrwxr-x 2 kali     comptia 4096 Aug 15 10:27 permissions
-rw-rw-rw- 1 kali     kali      44 Aug 15 10:28 user.txt
```

- The **chown comptia: linux.py** command only changed the owner as *comptia*, not the group.

```
┌──(root㉿kali)-[/home/kali/Desktop/comptia_linux+_exam]
└─# chown kali:kali permissions/

┌──(root㉿kali)-[/home/kali/Desktop/comptia_linux+_exam]
└─# ls -l
total 16
-rw-r-xr-- 1 kali    comptia   22 Aug 15 10:31 executable.py
-rw-r-x--- 1 comptia comptia   57 Aug 15 10:27 linux.py
drwxrwxr-x 2 kali    kali    4096 Aug 15 10:27 permissions
-rw-rw-rw- 1 kali    kali      44 Aug 15 10:28 user.txt
```

- The **chown kali:kali permissions** command changed both the owner and the group as *kali*.

Special Permissions and Attributes

- **Special Permission:** The *less privileged users* are allowed to execute a file by assuming the privileges of the file's owner or group. Special permissions are used to control how files and directories can be accessed and manipulated by users and processes.

-There are three main special permissions that can be used. They are **SUID (Set User ID)**, **SGID (Set Group ID)** and **Sticky Bit**.

1. **SUID:** *User* is allowed to have *similar permissions as the owner* of the file. When a file has the SUID bit set, *any user* who runs that file will *have the file executed with the permissions of the file's owner*, not the user who ran the file.

2. **GUID:** *User* is allowed to have *similar permissions as the group owner* of the files and directories. It provides the *permissions of the file's group to any user* who runs a file with GUID bit set.
3. **Sticky Bit:** Commonly used on directories like **/tmp** to *prevent users from deleting each other's files*. When it's set on a directory, it *restricts the ability to delete* or *rename files* within that directory to the file's owner, directory's owner or the root (super user).

-To give special permission, we use the **chmod** command in either *symbolic* or *absolute* mode. Additionally, we can determine the special permissions on files or directories by using **ls -la** command.

- Setting **SUID**:

  -When using the symbolic mode, **chmod u+s <filename>** command will be used.

  -When using the absolute mode, **chmod 4777 <filename>** command will be used (replace 777 as how you want to set).

- Setting **GUID**:

  -When using the symbolic mode, **chmod g+s <directory_name>** command will be used.

  -When using the absolute mode, **chmod 2777 <directory_name>** command will be used (replace 777 as how you want to set).

- Setting **Sticky Bit**:

  -When using the symbolic mode, **chmod +t <directory_name>** command will be used.

  -When using the absolute mode, **chmod 1777 <directory_name>** command will be used (replace 777 as how you want to set).

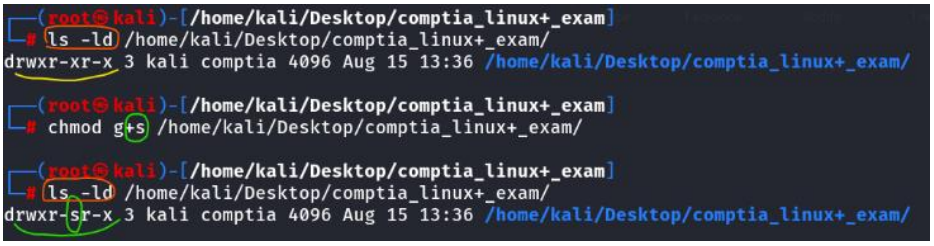
-To **remove** the special permissions, use the **minus** (-) operator in *symbolic* mode, or set to **0** in *absolute* mode.

-Files can have one or more attributes set that define how the system interacts with them.

- **Immutable Flag:** It's an attribute of a file or directory that *prevents it from being modified*. In other words, **no one** can *delete*, *rename* or *write* to an immutable file.
- **lsattr:** This command is used to *list the attributes* of a file or directory. It has several options. Such as, **-R** option lists attributes recursively, **-a** option lists all files, **-d** option lists directories, **-v** option lists file's version number.
- **chattr:** This command is used to *change the attributes* of a file or directory. It has several options. Such as, **-R** option changes attributes of directories and content recursively, **-v** option sets file's version number, **+I** option marks file as read-only and immutable, **-I** option removes the read-only permission attribute.
- **Access Control List (ACL):** It enables a more granular level of control than simply using file permissions.
- **getfacl:** This command displays the *metadata about the items including its owner, group,* any *SUID, SGID* or *sticky bit flags* that are set, the standard permissions associated with the item and the individual permission entries for users and groups. It's useful when retrieving the ACLs of files and directories.
- **setfacl:** This command is used to *change the permissions associated with the ACL* of a file or directory. It has several options. Such as, **-r** option sets ACL options recursively, **-s** option sets or replaces ACL, **-m** option modifies the existing ACL of an item, **-x** option removes entries from existing ACL, **-b** option removes all entries except standard permissions.

Hands-on Experience – ls -ld, chmod, chattr, lsattr, getfacl, setacl

-Let's *set* **GUID** for a directory.



- In this case, as we can see, after executing the changes, there is '**s**' letter which represents the GUID. Keep in mind that, the **chmod 2755**

/home/kali/Desktop/comptia_linux+_exam command should also be worked instead of the symbolic mode usage.

-Let's *implement* **sticky bit**. It will protect files from deletion by anyone other than the owner and root.

```
┌──(root㉿kali)-[/home/kali/Desktop/comptia_linux+_exam]
└─# chmod +t /home/kali/Desktop/comptia_linux+_exam/

┌──(root㉿kali)-[/home/kali/Desktop/comptia_linux+_exam]
└─# ls -ld /home/kali/Desktop/comptia_linux+_exam/
drwxr-sr-t 3 kali comptia 4096 Aug 15 13:36 /home/kali/Desktop/comptia_linux+_exam/
```

- In this case, as we can see, after executing the changes, there is '**t**' letter which represents the sticky bit. Keep in mind that, the **chmod 1755 /home/kali/Desktop/comptia_linux+_exam** command could also be used instead of the symbolic mode usage.

-Let's implement **immutable flag** to a file.

```
┌──(root㉿kali)-[/home/kali/Desktop/comptia_linux+_exam]
└─# chattr +i README

┌──(root㉿kali)-[/home/kali/Desktop/comptia_linux+_exam]
└─# lsattr README
----i---------e------- README

┌──(root㉿kali)-[/home/kali/Desktop/comptia_linux+_exam]
└─# rm README
rm: cannot remove 'README': Operation not permitted

┌──(root㉿kali)-[/home/kali/Desktop/comptia_linux+_exam]
└─# rm -rf README
rm: cannot remove 'README': Operation not permitted
```

- As we can see, even root cannot remove the file.

-We can *remove* the **immutable flag** by using **chattr** command with **-i** option.

```
┌──(root㉿kali)-[/home/kali/Desktop/comptia_linux+_exam]
└─# chattr -i README

┌──(root㉿kali)-[/home/kali/Desktop/comptia_linux+_exam]
└─# lsattr README
--------------e------- README

┌──(root㉿kali)-[/home/kali/Desktop/comptia_linux+_exam]
└─# rm README
```

-We can *retrieve information* about **ACL** by using **getfacl** command.

```
┌──(root㉿kali)-[/home/kali/Desktop/comptia_linux+_exam]
└─# getfacl /home/kali/Desktop/comptia_linux+_exam/
getfacl: Removing leading '/' from absolute path names
# file: home/kali/Desktop/comptia_linux+_exam/
# owner: comptia
# group: comptia
# flags: --t
user::rwx
group::r-x
other::r-x
```

-We can *modify* the **ACL** by using **setfacl** command.

```
┌──(root㉿kali)-[/home/kali/Desktop/comptia_linux+_exam]
└─# setfacl -R -m g:kali:r /home/kali/Desktop/comptia_linux+_exam/

┌──(root㉿kali)-[/home/kali/Desktop/comptia_linux+_exam]
└─# getfacl /home/kali/Desktop/comptia_linux+_exam/
getfacl: Removing leading '/' from absolute path names
# file: home/kali/Desktop/comptia_linux+_exam/
# owner: comptia
# group: comptia
# flags: --t
user::rwx
group::r-x
group:kali:r--
mask::r-x
other::r-x
```

- In this case, **-R** option applies to the *ACL changes recursively*, **-m** option specifies *ACL modification operation*, which is the *kali group have read permission*. In the output of second command, we can notice that any member in *kali group* can *only read* the contents of this directory.

Troubleshooting Permissions Issues

-The process of troubleshooting usually begins with the *identification of a problem* and ends with *restored services*. Troubleshooting aims to solve a problem efficiently with a minimal interruption of service.

-The *CompTIA troubleshooting model* contains **seven steps**:

1. Identify the problem.
2. Establish theory of probable cause.
3. Test the theory to determine the cause.
4. Establish an action plan.
5. Implement the solution.
6. Verify full system functionality.
7. Document findings, actions, and outcomes.

-Use the **ls -la** command to *verify permissions* and *verify the user* and *group ownership* of a file or directory.

-Use the **group <username>** command to *discover what groups a user is member of*.

-Use the **usermod** command to *change group membership*.

-The **lid** and **libuser-lid** commands can be used to *retrieve all members of a group*, including members whose primary group is the group being searched for.

- **getent:** This command enables to *retrieve group members of non-standard authentication methods*.

Storage Unit in Linux

File Systems

-There are many types of storage devices that are supported by the Linux operating system. Common types are *Hard Disk Drives*, *Solid-State Devices*, *USB Thumb Drives*, *External Storage Drives*. In Linux, devices are referred to as either **Block** or **Character** Devices.

1. ***Block Devices:*** Devices that *have read/write in blocks of data*. Basically, they are things like *storage devices* (HDD, SSD etc).
2. ***Character Devices:*** Devices that have *read/write in character streams of data*. Basically, they are things like *keyboards*, *mice*, *serial ports* etc.

- ***File System:*** A data structure used by an operating system to *store*, *retrieve*, *organize*, and *manage* files and directories *on storage devices*. Supported Linux File System:
    1. ***FAT (File Allocation Table):*** An *older file system* compatible with different operating systems.
    2. ***ext2:*** It used to be the *native Linux file system* of some older releases.
    3. ***ext3:*** It's a more *modern version of a native Linux file system*. It's faster in *recovering data* and better in *ensuring data integrity* in abrupt system shutdowns.
    4. ***ext4:*** It's one another *default file system in Linux* distributions. It supports volumes up to *one exabyte* and files up to *16 terabytes* in size.
    5. ***XFS:*** It's a 64-bit, *high-performance* journaling file system that provides *fast recovery* and can *handle large files* efficiently.
    6. ***BTRFS:*** It supports volumes of up to *16 exabytes* in size and up to *18 quintillion* files on each volume.

-Some file systems act like *protocols* so they can **share data over a network**. For example, *SMB* (*Server Message Protocol*) Protocol, *CIFS* (*Common Internet File System*), *NFS* (*Network File System*). Key features of these protocols include:

1. ***SMB:***
    - Allows user to have access to files over the **LAN**.

- **Server-client** architecture.
- Better for mixture of Windows and Linux file systems.
- Microsoft Windows supports SMB by default.

2. ***CIFS:***
   - Some Linux distributions refer to SMB as CIFS.
   - Rarely in use today.
3. ***NFS:***
   - Better for all Linux networks and file systems.
   - **Not supported** by Microsoft Windows by default.

- **Index Node (Inode):** Stores **metadata** about a file or directory on a file system such as *time-based values* (creation-modification date etc.), *permissions* and *ownership information*. Index nodes can be determined by using **df -i <path>** command.
- **Virtual File System (VFS):** File system created as interface between kernel and read file system. It translates real file system details over the kernel. We can mount many file systems on the same Linux installation, and they will appear uniform to system and applications. File system labels are used for easy identification, and many be up to 16 characters long. Labels can be displayed or changed using the following commands:
  - **e2label:** For *ext-based* file systems.
  - **xfs_admin:** For *XFS-based* file systems.

Partitioning

- **Partition:** A section of the storage drive that *logically acts as a separate drive*. It enables us to divide our drive into pieces, to manage the data more easily. There are mainly three types of partitions:
  1. ***Primary:*** Sometimes referred to as a *volume* and contains one file system or logical drive. *Swap file system* and *Boot partition* are normally created in this partition.
  2. ***Extended:*** Referred to as *logical drives* and contains several file systems. This partition does not contain any data.
  3. ***Logical:*** Partitioned and allocated as an *independent unit* and functions as a *separate drive*.

-When we need to create a new partition table or modify existing entries on the partition table, we will use **fdisk** utility.

- **fdisk:** This utility is used to *create, modify, or delete partitions* on a storage device. It has several options. Such as, **-b** option specifies number of drive sectors, **-H** option specifies number of drive heads, **-S** option specifies number of sectors per track, **-s** option prints partition size in blocks, **-l** option lists partition tables for devices.
- **parted:** This utility is also used to *create, destroy, and resize partitions* and runs the GNU Parted utility. The parted includes a menu-driven interactive mode just like fdisk. Some of the options are, **select** option chooses specified device or partition to modify, **mkpart** option creates partition with file system type specified, **print** option lists partition table, **resizepart** option resizes or modifies a partition's end position, **rm** option deletes a partition, **quit** option terminates the GNU parted utility.

-After creating a partition, we can't add a file system to that partition unless the kernel can read it from the partition table. Instead of rebooting, we can use **partprobe** command.

- **partprobe:** This command is used to *update the kernel* if there are any changes within the partition table.
- **mkfs:** This command stands for '*make file system*'. It's used to build a Linux file system on a device, which is usually a drive partition. It has several options. Such as, **-v** option produces verbose output that keeps changing as the program processes, **-V** option produces output which includes all file system-specific commands executed, **-t** option specifies type of file system to build, **-fs** option passes file system-specific options to the file system builder, **-c** option checks the device for bad blocks before building the file system, **-l** option reads the list of bad blocks from a specified file.
- **/etc/fstab file:** This file stores information about *storage devices*, *partitions* and where and how they should be *mounted*. It's getting read by the system during the boot up process. It can only be edited by *root*. Each line of this file has six fields that are going to be separated by spaces or tabs.
    1. ***Device/Partition Name:*** Name of the device or file system to mount.
    2. ***Default Mount Point:*** Where the file system is to be mounted.

3. ***File System Type:*** Type of file system used.
4. ***Mount Options:*** Set of comma-separated options that will be activated when the file system is mounted.
5. ***Dump Options:*** Indicates if the dump utility should back up the file system.
6. ***fsck Options:*** Order in which the fsck utility should check file systems.

- **/etc/crypttab file:** This file stores information about *encrypted devices* and *partitions* that must be unlocked and mounted on system boot.
- **/dev/ directory:** This directory contains details about all the files and subdirectories housed within it. Everything in Linux is *treated as a file*, even file systems.

-Linux uses naming conventions so that storage devices are easily located by the system whenever they are attached and when the system boots up.

- **/dev/sda1:** It's a controller-based naming. '*sd*' portion refers to the *type of controller*. 'a' portion refers to the *first whole drive* since a is the first letter of the alphabet. '*1*' refers to the *first partition on this drive*.
- **/dev/disk/by-id:** It's a *persistent naming scheme*. Device's *hardware serial number*.
- **/dev/disk/by-path:** It's also a *persistent naming scheme*. *Shortest physical path to the device*.
- **/dev/disk/by-uuid:** It's also a persistent naming scheme. *Universally unique identifier* (UUID).

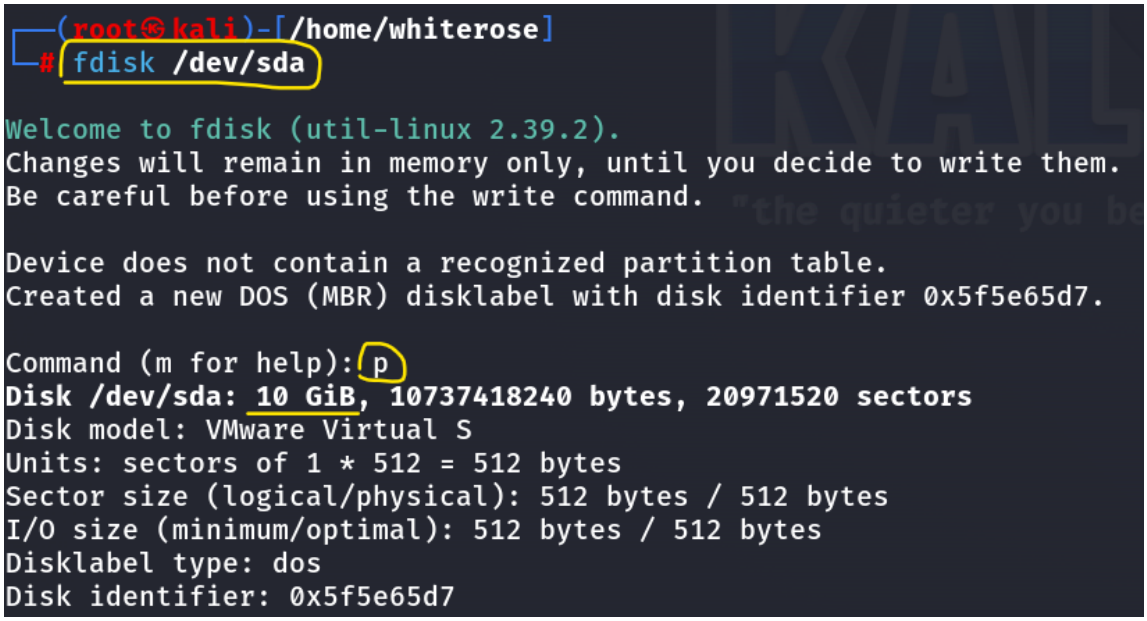
-There are also some special character devices inside of the /dev/ directory.

- **/dev/null:** A special type of virtual device that *discards anything* being sent or redirected into it. It's also known as ***black hole***. When we want to read from /dev/null, it's going to return to the *End of File*.
- **/dev/zero:** A special type of virtual device that *returns a null character* anytime you read from it. The difference between /dev/zero and /dev/null is when we want to read from /dev/zero, it's going to send *ASCII null character of 0x00*. It's useful when we're trying to *sanitize a hard drive*, and to do this we use a command in format of: **dd if=/dev/zero of=dev/sda1 bs=1GB count=1024** (overwrite a 1 TB HDD which is mounted as sda1).

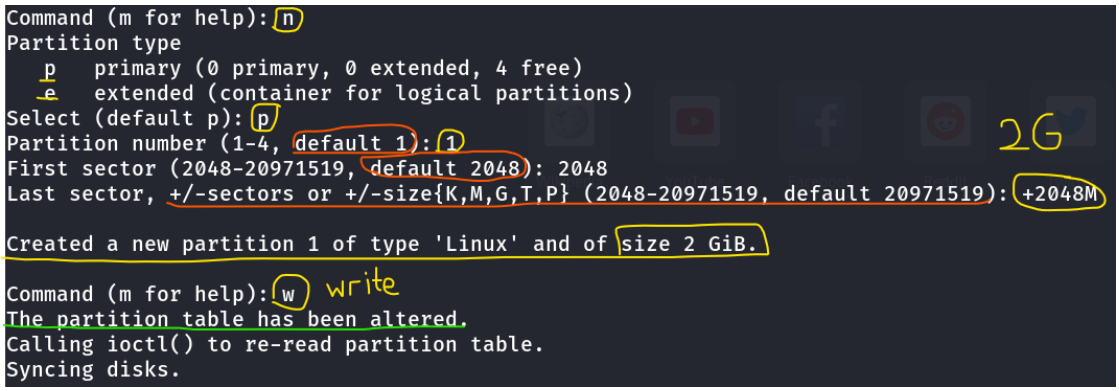
- **/dev/urandom:** A special type of virtual device that returns a *randomized series of pseudorandom*.

Creating Partitions – fdisk, partprobe, mkfs, parted, xfs_admin, e2label

-Let's see how we can partition a file system in Linux. Before we delve into that, make sure that an additional drive is added to virtual machine (**/dev/sda** in this case).



-We can get into fdisk utility's interface by using **fdisk <disk_name>** command (help page can be seen with **m** option).



- In this case, disk was partitioned using different options. The **n** option is used to create a new partition, **p** option under the **n** option represents

primary. Partition is set as 2048 million bytes which is 2 Gigabyte. After creating the partition, **w** option *writes changes* to disk.

-After we're done with fdisk, we have to update the kernel. When doing this, we can *reboot the system* or use a command called **partprobe**.

```
┌──(root㉿kali)-[/home/whiterose]
└─# partprobe /dev/sda
```

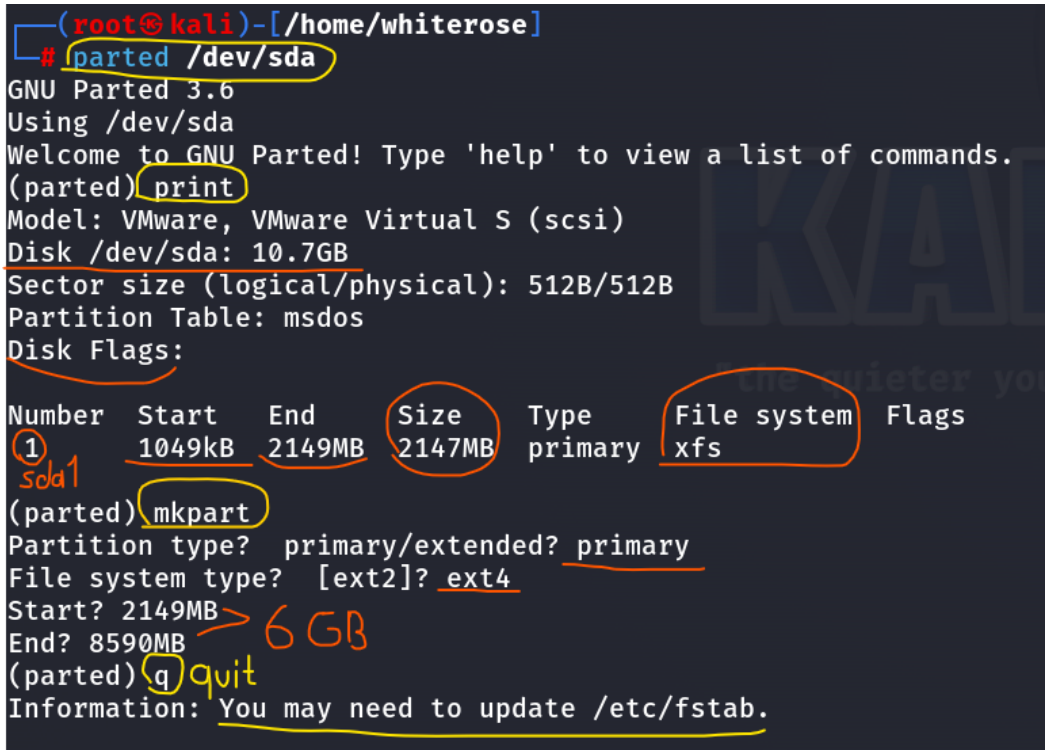
- In this case, the partitioned disk is /dev/sda.

-We can use **mkfs** command to set the file system of a partition. For setting file system as XFS, we will be using **mkfs.xfs** command.

```
┌──(root㉿kali)-[/home/whiterose]
└─# mkfs.xfs /dev/sda1
meta-data=/dev/sda1              isize=512    agcount=4, agsize=131072 blks
         =                       sectsz=512   attr=2, projid32bit=1
         =                       crc=1        finobt=1, sparse=1, rmapbt=1
         =                       reflink=1    bigtime=1 inobtcount=1 nrext64=1
data     =                       bsize=4096   blocks=524288, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0, ftype=1
log      =internal log           bsize=4096   blocks=16384, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
```

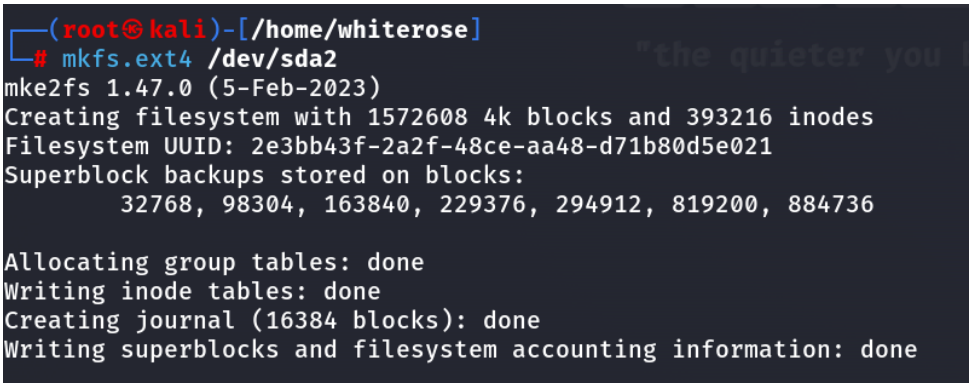
- In this case, the file system was set as XFS on new partition.

-We can also create a partition on disk by using **parted** command. It's a *GNU parted* tool.



- In this case, a new partition is created on the newly mounted disk, which is 6 GB size. The **mkpart** subcommand allowed to *create a new partition*. Since it's the second partition on the /dev/sda disk, it will be named as **sda2** (previous one was sda1).

-We can set the file system of a partition as ext4 by using **mkfs.ext4** command.



-When displaying the label of partitions that use XFS file system, we will use **xfs_admin** command. We can display label name with **-l** option.

```
┌──(root㉿kali)-[/home/whiterose]
└─# xfs_admin -l /dev/sda1
label = ""

┌──(root㉿kali)-[/home/whiterose]
└─# xfs_admin -l /dev/sda2
xfs_admin: /dev/sda2 is not a valid XFS filesystem (unexpected SB magic number 0x00000000)
Use -F to force a read attempt.
```

- As you can see on the output of second command, there was *error* while attempting to list the label of sda2 disk since the sda2 partition set as *ext4*. However, since *sda1* uses *XFS* file system, it has *correctly* displayed.

-We can set label for partitions that use XFS file system by using **xfs_admin** command with **-L** option.

```
┌──(root㉿kali)-[/home/whiterose]
└─# xfs_admin -L Backup /dev/sda1
writing all SBs
new label = "Backup"

┌──(root㉿kali)-[/home/whiterose]
└─# xfs_admin -l /dev/sda1
label = "Backup"
```

- In this case, label of sda1 partition was being set as ‘Backup’.

-To display or set the label of partitions that use ext file systems (ext2, ext3, ext4), we will be using **e2label** command.

```
┌──(root㉿kali)-[/home/whiterose]
└─# e2label /dev/sda2
```

- In this case, no argument is provided, which displays the label (empty).

```
┌──(root㉿kali)-[/home/whiterose]
└─# e2label /dev/sda2 PhotoBackup

┌──(root㉿kali)-[/home/whiterose]
└─# e2label /dev/sda2
PhotoBackup
```

- As we can see, we only need to provide a name as argument while setting the label.

Logical Volumes

-In Linux, the **Device Mapper** *creates virtual device* and *passes data* from that virtual device to one or more physical device.

-**DM-Multipath** is a feature of the Linux kernel that provides *redundancy* and *improved performance* for *block storage devices*. If one path fails, DM-Multipath will switch to one of the other available paths. Typically, the multipath tools package configuration file can be found in **/etc/multipath.conf**.

- **mdadm:** This command is a tool that is used to *manage software-based RAID arrays*. In RAID array, data is stored across multiple physical storage devices and those devices are combined into a single virtual storage device. This type of software-based RAID configuration is an *alternative* to using *device mapper* and the *DM-Multipath*.
- **RAID (Redundant Array of Inexpensive Disks):** RAID is a technique that makes use of a *combination of multiple disks* for storing the data instead of using a single disk for increased performance, *data redundancy*, or to protect data in case of a drive failure. It's also known as *Redundant Array of Independent Disks*. RAIDs have a couple of types and each of them is used for specific needs. Such as *RAID 0*, *RAID 1*, *RAID 5*, *RAID 6*, *RAID 10*. Each type of RAID has either striping, mirroring, parity or a combination of these features depending on its type.
    - **Striping:** It combines multiple smaller physical disks to logically act as a single larger disk. For instance, imagine that we need 4 TB HDD for storage, but we only have two pieces of 2 TB HDDs

available, so we can use striping to combine these two pieces of 2 TB HDDs logically.

- **Mirroring:** It combines two physical hard drives into a single logical volume where an identical copy of everything is put on both drives. So, if a drive fails, the system can quickly access all the files it needs, because there's still a fully redundant copy on that second drive as part of that RAID.
- **Parity:** It is used in RAID drive arrays for fault tolerance by calculating the data in two drives and storing the results on a different drive.

-RAID has several types. Each of them has its own unique characteristics. Let's define them.

1. **RAID 0:** It relies on *striping*. We have *two different disks* and each of them holds *half of the data*. It's used when we need a *speed boost*. There is *no loss of disk space*. However, **NO data redundancy**.
2. **RAID 1:** It relies on *mirroring*; it means disks have the *same data*. It's **fully redundant** in case of any troubles. It has a negative side which is the *loss of space on one of those disks*. For instance, imagine that we have *two 1 TB HDD*, we *no longer get 2 TB storage space* to use, instead we're getting *only 1 TB* since one of the disks has the *copy of every single piece of data*.
3. **RAID 5:** Most popular one. It relies on both *striping* and *parity*. A RAID 5 is going to use *striping* and a *computed parity* across three or more disks. Because the *parity data is spread* across all drives, RAID 5 is considered one of the *most secure* RAID types. RAID 5 can use *three* (**at least**) or *more disks*. It's **data redundant**. It can *lose one disk* and can *still calculate based on the remaining two disks*. It's *more efficient* in terms of space compared to RAID 1.
4. **RAID 6:** Everything that was true about RAID 5 is also valid for RAID 6 except in RAID 6, there is *double parity* instead of single parity. So, RAID 6 requires *at least 4 disks* since it relies on double parity. Advantage of RAID 6, we can *still operate* even though we *lose two disks*.

5. **RAID 10:** RAID 10 is '*RAID of RAIDS*'. It relies on *mirroring* and *striping*. Requires *at least 4 disks*. If we have four disks, we have *two fully redundant mirror disks*, so we *lose half of our storage space*. RAID 10 is considered to be *fast* because of the *RAID 0's striping technique* across two mirrored arrays. Basically, we have the *benefits of RAID 0 and RAID 1* while using RAID 10.

-If we want to get information about RAID configuration inside of our Linux system and its kernel, we can take a look at the **/proc/mdstat** file.

- **/proc/mdstat file:** This file contains a snapshot of the kernel's RAID/md state.
- **Logical Volume Manager (LVM):** LVM maps whole physical devices and partitions into one or more virtual containers called *volume groups*. With Logical Volume Manager, we can dynamically *create*, *resize* and *delete* volumes without rebooting the system. We can also map logical volumes across physical devices. Additionally, we can create virtual snapshots of each logical volume.

-The **/dev/mapper/** directory contains all the logical volumes on a given system that are being managed by LVM.

-The Logical Volume Manager divides its volume management tools into three categories, including **Physical Volume Tools**, **Volume Group Tools**, **Logical Volume Tools**.

1. **Physical Volume Tools:**
   - ***pvscan:*** Scans for all physical devices being used as physical volumes.
   - ***pvcreate:*** Initializes a drive or partition to use as a physical volume.
   - ***pvdisplay:*** Lists attributes of physical volumes.
   - ***pvchange:*** Changes attributes of a physical volume.
   - ***pvs:*** Displays information about physical volumes.
   - ***pvck:*** Checks the metadata of physical volumes.
   - ***pvremove:*** Removes physical volumes.
2. **Volume Group Tools:**

- ***vgscan:*** Scans all physical devices for volume groups.
- ***vgcreate:*** Creates volume groups.
- ***vgdisplay:*** Lists attributes of volume groups.
- ***vgchange:*** Changes attributes of volume groups.
- ***vgs:*** Displays information about volume groups.
- ***vgck:*** Checks the metadata of volume groups.
- ***vgrename:*** Renames a volume group.
- ***vgreduce:*** Removes physical volumes from a group to reduce its size.
- ***vgextend:*** Adds physical volumes to volume groups.
- ***vgmerge:*** Merges two volume groups.
- ***vgsplit:*** Splits a volume group into two.
- ***vgremove:*** Removes volume groups.

3. **Logical Volume Tools:**
   - ***lvscan:*** Scans all physical devices for logical volumes.
   - ***lvcreate:*** Creates logical volumes in a volume group.
   - ***lvdisplay:*** List attributes of logical volumes.
   - ***lvchange:*** Changes attributes of the volumes.
   - ***lvs:*** Displays information about logical volumes.
   - ***lvrename:*** Renames logical volumes.
   - ***lvreduce:*** Reduces the size of logical volumes.
   - ***lvextend:*** Extends the size of logical volumes.
   - ***lvresize:*** Resizes logical volumes.
   - ***lvremove:*** Removes logical volumes.

Managing Logical Volumes – pvcreate, pvdisplay, vgcreate, vgdisplay, vgscan, lvscan, lvextend, lvreduce

-Linux uses device mapping for managing logical volumes. Device mapping creates virtual devices and passes data from that virtual device to one or more physical devices. A major application of the device mapper is the **Logical Volume Manager**.

-To see the logical volumes on the system, we need to take a look at the **/dev/mapper/** directory.

```
┌──(root㉿kali)-[~whiterose/Desktop]
└─# ls /dev/mapper
control
```

-We can create physical volumes by using **pvcreate** command.

```
┌──(root㉿kali)-[/home/whiterose]
└─# pvcreate /dev/sda1 /dev/sda2
WARNING: xfs signature detected on /dev/sda1 at offset 0. Wipe it? [y/n]: y
  Wiping xfs signature on /dev/sda1.
  Physical volume "/dev/sda1" successfully created.
  Physical volume "/dev/sda2" successfully created.
```

- In this case, Physical Volumes are **/dev/sda1** and **/dev/sda2**.

-We can display the physical volumes by using **pvdisplay** command.

```
┌──(root㉿kali)-[/home/whiterose]
└─# pvdisplay
  "/dev/sda1" is a new physical volume of "2.00 GiB"
  --- NEW Physical volume ---
  PV Name               /dev/sda1
  VG Name
  PV Size               2.00 GiB
  Allocatable           NO
  PE Size               0
  Total PE              0
  Free PE               0
  Allocated PE          0
  PV UUID               vPn2XV-XGv8-rHLV-wy4U-Ift0-vGLP-Ya6udc

  "/dev/sda2" is a new physical volume of "2.00 GiB"
  --- NEW Physical volume ---
  PV Name               /dev/sda2
  VG Name
  PV Size               2.00 GiB
  Allocatable           NO
  PE Size               0
  Total PE              0
  Free PE               0
  Allocated PE          0
  PV UUID               kISYla-0Gce-DKfF-sCBf-Wtqn-P2Ec-8LtTRj
```

-We can create a volume group from the physical volumes created previously by using **vgcreate** command.

```
┌──(root㉿kali)-[/home/whiterose]
└─# vgcreate extra /dev/sda1 /dev/sda2
  Volume group "extra" successfully created
```

- In this case, as physical volumes; */dev/sda1* and */dev/sda2* are used. And the name of the Volume Group is *extra*.

-We can display the volume groups on the system by using **vgdisplay** or **vgscan** command.

```
┌──(root㉿kali)-[/home/whiterose]
└─# vgscan
  Found volume group "extra" using metadata type lvm2

┌──(root㉿kali)-[/home/whiterose]
└─# vgdisplay
  --- Volume group ---
  VG Name               extra
  System ID
  Format                lvm2
  Metadata Areas        2
  Metadata Sequence No  1
  VG Access             read/write
  VG Status             resizable
  MAX LV                0
  Cur LV                0
  Open LV               0
  Max PV                0
  Cur PV                2
  Act PV                2
  VG Size               3.99 GiB
  PE Size               4.00 MiB
  Total PE              1022
  Alloc PE / Size       0 / 0
  Free  PE / Size       1022 / 3.99 GiB
  VG UUID               pmX2kb-JLLe-vf6S-j9lm-Ig3E-fsX2-78Vbka
```

-We can create logical volume and associate it with the volume group by using **lvcreate** command.

```
┌──(root㉿kali)-[/home/whiterose]
└─# lvcreate --name additional --size 1GB extra
  Logical volume "additional" created.
```

- In this case, the name of the logical volume is *additional*, and the associated volume group is *extra*. The size of the logical volume is 1 GB.

-We can scan and display physical devices for logical volumes by using **lvscan** command.

```
┌──(root㉿kali)-[/home/whiterose]
└─# lvscan
  ACTIVE            '/dev/extra/additional' [1.00 GiB] inherit
  ACTIVE            '/dev/extra/tools' [2.00 GiB] inherit
```

- In this case, there are two logical volumes: *additional* and *tools*.

-We can extend or reduce the size of the logical volume by using **lvextend** and **lvreduce** commands.

```
┌──(root㉿kali)-[/home/whiterose]
└─# lvreduce -L1G /dev/extra/tools
  No file system found on /dev/extra/tools.
  Size of logical volume extra/tools changed from 2.00 GiB (512 extents) to 1.00 GiB (256 extents).
  Logical volume extra/tools successfully resized.
```

```
┌──(root㉿kali)-[/home/whiterose]
└─# lvextend -L2G /dev/extra/additional
  Size of logical volume extra/additional changed from 1.00 GiB (256 extents) to 2.00 GiB (512 extents).
  Logical volume extra/additional successfully resized.
```

- In this case, '*tools*' logical volume has reduced to 1 GB from 2GB, '*additional*' logical volume has extended to 2 GB from 1 GB. We can verify the changes by using **lvscan** or **lvdisplay** commands.

Mounting File Systems

- **Mount Point:** An access point that is typically an empty directory where a *file system* is *loaded* or *mounted* to make it accessible to users.
- **mount:** This command is used to *load a file system to a specified directory*. By mounting a file system to a directory, we can make it accessible to users and applications. It has several subcommands. Such as, **auto** specifies that devices has to be mounted automatically, **noauto** specifies that devices should not be mounted automatically, **nouser** specifies that only the root user can mount a device or a file system, **user** specifies that all users can mount a device or a file system, **exec** allows binaries in a file system to be executed, **noexec** prevents binaries in a file system from being executed, **ro** mounts file system as read-only, **rw** mounts file system with read-write permissions, **sync** specifies that input and output operations should be done

synchronously, **async** specifies that input and output operations should be done asynchronously.

- **umount:** This command stands for unmount. It's used to *unmount a mounted file system*. Key thing is that file system we wish to unmount *must not be in use* currently. It has several options. Such as, **-l** option performs a lazy unmount, **-R** option recursively unmount specified directory mount points, **-t** option unmounts only the file system types specified, **-O** option unmounts only the file systems with specified options in the /etc/fstab file, **-fake** option tests the unmounting procedure.
- **fstab (File System Table):** A list of *file systems to be mounted*, *their mount points*, and any options that might be needed for specific file systems.

-Another way to mount a file system is to use the **systemd** or **system daemon**. The **systemd.mount** can be used to *create a new mount unit* to mount the file system. To set this up, we need to create a text file under the **/etc/systemd/system** directory and name it as **random.mount**.

- **Filesystem in Userspace (FUSE):** It is a software interface in Linux that lets *non-privileged users* create their *own file systems* without editing the underlying kernel code. It allows to specify how read, write, and stat requests can be made to a file system.

Hands-on Experience – mkdir -p, mount, lvscan, /dev/mapper

-We are going to associate the logical volumes to a directory that we're going to create now by using **mkdir** and **mount** commands.

```
┌──(root㉿kali)-[/]
└─# mkdir -p /Additional/Tools /Additional/PDFs

┌──(root㉿kali)-[/]
└─# mount /dev/extra/additional /Additional/PDFs

┌──(root㉿kali)-[/]
└─# mount /dev/extra/tools  /Additional/Tools
```

```
┌──(root㉿kali)-[/]
└─# lvscan
  ACTIVE            '/dev/extra/additional' [2.00 GiB] inherit
  ACTIVE            '/dev/extra/tools' [1.00 GiB] inherit
```

- In this case, these two logical volumes are being used.

-We can verify the status of mounting process by listing the **/dev/mapper/** directory.

```
┌──(root㉿kali)-[/]
└─# ls -la /dev/mapper
total 0
drwxr-xr-x  2 root root     100 Aug 23 10:15 .
drwxr-xr-x 19 root root    3540 Aug 23 10:15 ..
crw-------  1 root root 10, 236 Aug 22 11:51 control
lrwxrwxrwx  1 root root       7 Aug 23 10:00 extra-additional -> ../dm-0
lrwxrwxrwx  1 root root       7 Aug 23 10:15 extra-tools -> ../dm-1
```

-Next thing we will do is to ensure the logical volumes are mounted on boot. To do this, we are going to edit **/etc/fstab** file.

```
/dev/extra/additional/ /Additional/PDFs ext4 defaults 0 0
/dev/extra/tools /Additional/Tools xfs defaults 0 0
```

- In this case, these two lines have been added according to its specs.

-As next thing, we will test if we can mount these directories with no errors by using **mount -a**.

```
┌──(root㉿kali)-[/home/whiterose]
└─# mount -a
```

Managing File Systems

- **/etc/mtab file:** Reports the status of *currently mounted file systems*.

-/etc/mtab file is similar to **/proc/mounts** file. However, /proc/mounts is more accurate and includes more up-to-date information on file systems.

- **/proc/partitions file:** Contains information about *each partition attached to the system*. It includes columns that include **major**, **minor**, number **blocks** and **name**. The **major** column represents the class of device, the **minor** column separates partitions into physical devices, the **number of blocks** column refers to the number of physical blocks and the **name** column refers to the name of the partition.
- **lsblk:** This command displays information about *block storage devices* currently available on the system. It has several options. Such as, **-a** option

lists empty devices, **-r** option lists devices excluding provided output devices, **-f** option displays additional information, **-l** option displays results in list format, **-m** option displays device permission information.

- **blkid:** This command prints each block device in a *flat format* and includes some additional information. Similar to lsblk command but it prints in flat format, *file system type* and *device partitioning UUID* additionally.

-Some of the most common tools for managing ONLY **ext** file systems:

- **e2fsck:** Used to check the correctness of ext file systems.
- **resize2fs:** Used to resize ext file systems.
- **tune2fs:** Used to adjust various tunable parameters of the ext2 and ext3 file systems.
- **dumpe2fs:** Used to print the superblock and block group information for ext file systems.

- **Superblock:** Contains *metadata about the file system*, including its size, type and status.
- **fsck:** This command is used to *check the correctness* and *validity of a file system*. Most systems execute fsck command *while booting* in order to detect and correct any errors.

-Some of the most common tools for managing ONLY **XFS** file systems:

- ***xfs_info:*** It displays details about the XFS file system.
- ***xfs_admin:*** It changes the parameters of an XFS file system.
- ***xfs_metadump:*** It copies the superblock metadata of the XFS file system to a file.
- ***xfs_growfs:*** It expands the XFS file system to fill the drive size.
- ***xfs_copy:*** It copies the contents of the XFS file system to another location.
- ***xfs_repair:*** It repairs/recovers a corrupt XFS file system.
- ***xfs_db:*** It debugs the XFS file system.

- **lsscsi**: This command is used to list information about *SCSI devices* connected to a Linux system.
- **fcstat**: This command interacts with and displays statistics of *Fibre Channel connected devices*.

Linux Directory Structure

-Linux file system consists of *regular files*, *directories*, *special files*, *links*, *domain sockets*, and *named pipes*.

1. **Regular Files:** Includes text files, executable files, input for programs, and output from programs.
2. **Directories:** Containers for other files.
3. **Special Files:** System files stored below the /dev directory.
4. **Links:** Make a file accessible in multiple parts of the system's file tree.
5. **Domain Sockets:** Provide inter-process networking that is protected by the file system's access control.
6. **Named Pipes:** Enable processes to communicate with each other without using network sockets.

- **file:** This command is used to *determine the type of file*.
- **Filesystem Hierarchy Standard (FHS):** Specifies files and directories by their *names* and *locations* on Linux system. The top-most directory in a Linux system is always going to be **root** (/) directory. Underneath the root directory, there are standardized sub directories such as /bin, /boot, /etc, /dev, /var, /usr etc.
  - **/bin/:** Contains *command binaries* or *executable programs* that are required for the system to boot and run. For example, **ls**, **cat**, **mv** etc.
  - **/sbin/:** Contains *system binaries* for system administrations. Such as, **fdisk**, **iptables** etc.
  - **/etc/:** Contains *configuration files* for the system. Such as, **resolv.conf**, **sudoers**, **shadow** etc.
  - **/home/:** Contains *user home directories*.
  - **/root/:** It is the *home directory of root user*.
  - **/var/:** Contains *variable data files* that are expected to change in size. For instance, **log**, **spool**, **lib** etc.
  - **/tmp/:** Contains *temporary files* that are created and deleted during the course of system operation.
  - **/usr/:** Contains *shareable*, *read-only data*. It has subdirectories like **/usr/bin**, **/usr/sbin**, **/usr/lib** etc.

-**/usr/bin/:** Contains *binaries* and *executable programs that can be executed by all users*.
-**/usr/local/:** Includes *custom-built applications*.
-**/usr/lib/:** Includes *object libraries* and *internal binaries* for executable programs.
-**/usr/lib64/:** Same as /usr/lib but it's *meant for 64-bit systems*.
-**/usr/share/:** Includes *read-only files* about the system.

- **/boot/:** Contains *files needed for the system to boot*. Such as kernel image and bootloader files.
- **/lib/:** Contains *shared libraries* and *kernel modules*.
- **/opt/:** It is used for *optional* or *user-installed software packages*.
- **/mnt/:** It is used to *mount external file systems temporarily*. Such as USB drive and external hard drive.
- **/media/:** Stores *mount points for removable media*. Such as CD-ROMs and floppy disks.
- **/dev/:** Contains *device files*.
- **/proc/:** Contains *virtual files* that provide information about the *system* and *running processes*.
- **/sys/:** Contains *virtual files* that provide information about the *system's hardware devices* and *drivers*.

-*Current working directory* is represented as a **single dot** (.). If we go one level above from current working directory, it's referred to as *parent directory*. It's represented as a **double dot** (..).

Troubleshooting Storage Issues

- **ulimit:** This command *limits the system resources for a user* in a Linux-based server. For example, if we apply **ulimit -n 100** command, this will put the limit of 100 maximal open files for particular user.
- **df & du commands:** These commands *facilitate storage space tracking*. The **df** command displays the device's storage space. The **du** command displays how a device is used.
- **I/O Scheduling:** This process *manages the order of input* and *output operations for block storage devices*. There are different types of schedulers,

and these are, **Deadline Scheduler**, **CFQ (Complete Fair Queuing) Scheduler** and **NOOP Scheduler**.

-We can set the scheduler by modifying **/sys/block/queue/scheduler** file. For example, **echo noop > /sys/block/queue/scheduler command** will set the scheduler as NOOP.

- **iostat:** This command *generates device* and *CPU usage reports of the system*. Provides information like transfers (I/O requests) per second, blocks both read and written per second, total number of blocks both read and written. We can use **-d** option to specify only the device information.
- **ioping:** This command *tests I/O latency in real-time* just like ping command for network testing and generates a report. Basically, it's useful while we're troubleshooting latency with storage device. It has several options. For example, **-c** option specifies the number of I/O requests to perform before stopping, **-i** option sets the time interval between I/O requests, **-t** option sets the minimum valid request time, **-T** option sets the maximum valid request time, **-s** option sets the size of requests.
- **Storage Quotas:** It *allocates storage space for users*. It prevents issues with users using excessive storage space. In Linux, there a couple of Quota Management commands, let's have a look at them:
  - ***quotacheck -cug:*** Creates quota database files for a file system and checks for user and group quotas.
  - ***edquota -u:*** Edits quotas for a specific user.
  - ***edquota -g:*** Edits quotas for a specific group.
  - ***setquota -u:*** Sets quotas for a specific user.
  - ***setquota -g:*** Sets quota for a specific group.

-If we are using XFS file system, we can use **xfs_admin** command to configure quotas.

-Quota reports consist of *name of the user/group, number of blocks being used, user's/group's storage hard limit, grace period, number of inodes used, soft limit on inodes, hard limit on inodes*.

-There are several commands for generating quota reports:

- ***repquota -a:*** Displays the reports for all file systems indicated as read-write with quotas in the mtab file.
- ***repquota -u:*** Displays the quota report for particular user.
- ***quota -uv:*** Displays the quota report for a particular user with verbose output.
- ***warnquota -u****:* Checks if users are not exceeding the allocated quota limit.
- ***warnquoata -g:*** Checks if groups are not exceeding the allocated quota limit.

-While troubleshooting a storage issue, we can follow those steps:

- Ensure that devices are *physically connected* and *powered on*.
- Verify *device recognition* by the system.
- Check *configuration files* for errors and reload them.
- Confirm *storage capacity* and *workload*.
- Use **partprobe** command to scan for *new storage devices* and *partitions*.

Files and Directories

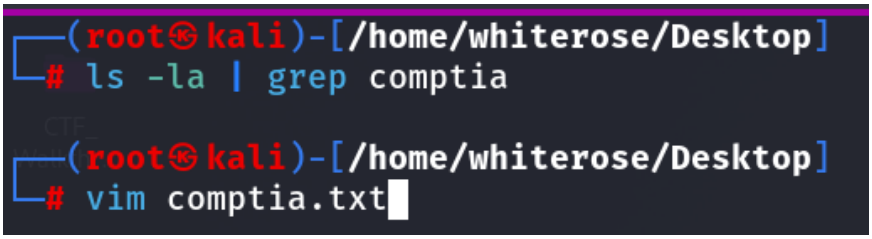
Create and Edit Text Files

- **Text Editor:** It's an application that enables users to *view*, *create*, or *modify the contents of text files*. Let's take a look at the text editors in Linux.
  - ***vi:*** Visual Text Editor originally created for Unix and was later cloned into open-source versions that are also run in Linux.
  - ***vim:*** It's default text editor in most Linux distributions.
  - ***Emacs:*** Flexible, powerful, and popular text editor used in Unix and Linux.
  - ***gVim:*** Graphical version of the vim editor.
  - ***gedit:*** Simple and powerful GUI-based text editor used in the GNOME desktop environment.
  - ***GNU nano:*** Small and user-friendly text editor.

- **vim:** It's the extended version of the vi editor. Vim can be invoked with **vim** and **vi** commands. Vim supports multiple files being opened simultaneously. Vim has modes, including **Insert**, **Execute**, **Command**, and **Visual Modes**. Some of the important arguments in Vim are **:w** (*to save*), **:q** (*to quit*), **:q!** (*to quit without saving*), **:qa** (*to quit all files*) **:wq** (*to save and quit*), **:e!** (*to revert to last saved format without closing*), **:! <Linux_command>** (*to execute the command*), **:help** (*to open help page*) etc. Navigation in Vim can be done with single-key shortcuts such as, **h** for *moving left one character*, **j** for *moving down one line*, **k** for *moving up one line*, **l** *for moving right one character*, **^** for *moving to the beginning of the current line*, **$** for *moving to the end of the current file* etc. There are also editing operators in Vim such as, **x** for *deleting the character selected by the cursor*, **d** for *deleting text*, **dd** for *deleting the current line*, **p** for *pasting text on the line below the cursor*, **P** for *pasting text on the line above the cursor* etc.
- **nano:** Nano editor is more *user-friendly* unlike vim. Nano can be invoked with **nano** command. Navigation and editing in nano are performed with shortcuts that use Ctrl key. These are **Ctrl + G** for *opening nano help page*, **Ctrl + X** for *exiting*, **Ctrl + O** for *saving file*, **Ctrl + J** for *justifying the current paragraph*, **Ctrl + R** for *inserting another file into the current file*, **Ctrl + W** for *searching in the file*, **Ctrl + K** for *cutting the currently selected line*, **Ctrl + U** for *pasting the line that was cut*, **Ctrl + C** for *displaying the cursor's position*, **Ctrl + V** for *navigating to the next page*, **Ctrl + Y** for *navigating to the previous page*.
- **gedit:** It's the default text editor for the GNOME desktop environment. Gedit has a GUI with a menu-based design. Similar to Notepad in Windows.

Hands-on Experience – vim, nano

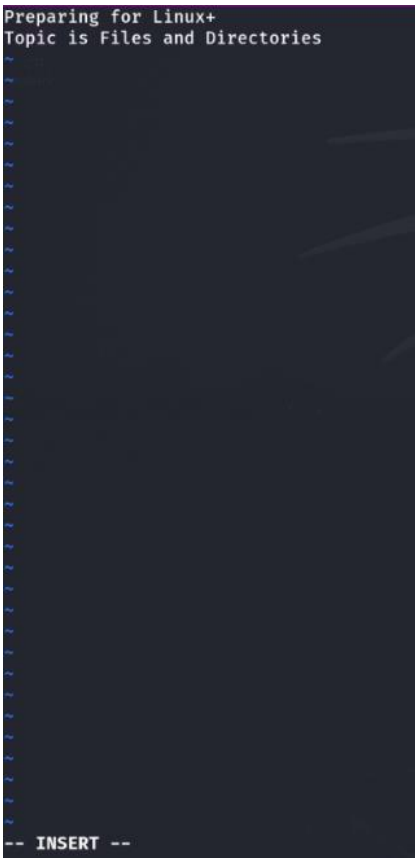
-We can create a file by using the text editor's name and file name. For example, **nano <file_name.txt>** or **vim <file_name.txt>**.

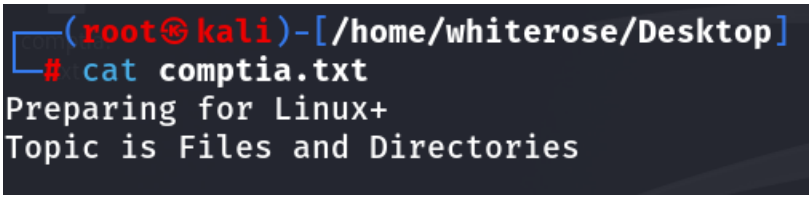


- As we can see, there is no file named *comptia*. It was created with vim.

-After creation of a file with vim, we need to switch to insert mode by using **I** key. There should be **-- INSERT --** banner at the bottom of the page.

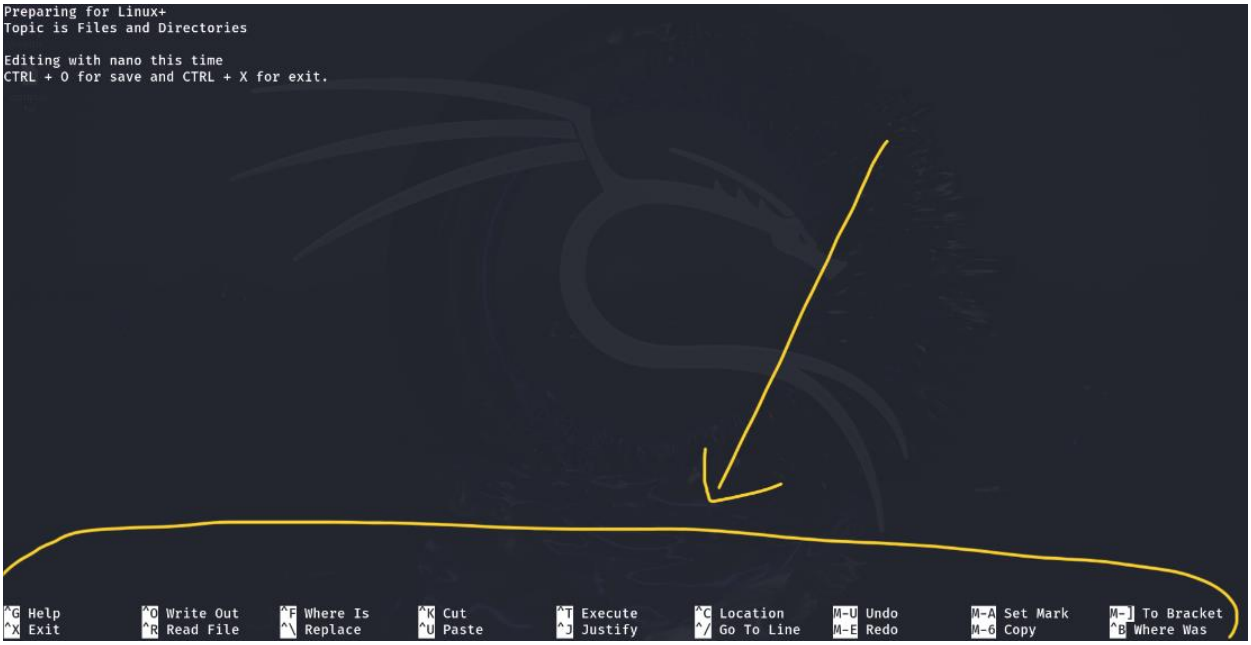
-We can cancel insert mode by using **Escape** key. And to save and exit, **:wq**. Colon for switching to execute mode, wq for writing changes and exiting.





- In this case, changes to the file have been saved after hitting **:wq**.

-Let's use **nano** editor.



- Key combinations have been specified at the bottom, shown within the circle. To save the changes to file, CTRL + O and CTRL + X for exiting the file.

Searching for Files

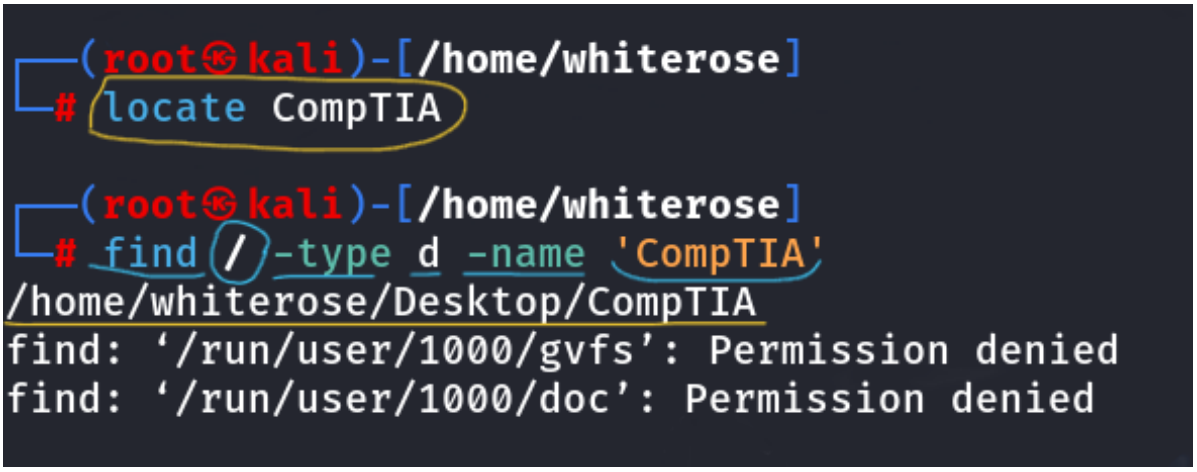
- **locate:** This command *performs a quick search for any specified file names* and *paths stored in the mlocate database*. This database must be updated regularly for the search to be effective. It has several options. Such as, **-r** option searches file names using regular expressions, **-c** option displays the number of matching entries found, -e option returns only files that exist at the time of search, **-I** option ignores the casing in file names or paths, **--n** option returns the first few matches up to the specified numbers.
- **updatedb:** This command *builds* and *updates* the *mlocate database* based on the */etc/update.conf* file.
- **find:** This command enables users to *search specific locations for files* and *directories* that adhere to search criteria. It has several options. For example, **-type** option to specify directory or file, **-name** option to specify the name of the item.

-Difference between locate and find command is that the locate command *searches from the database* whereas find command *performs a live search*. Since locate command searches from database, it's faster. However, if the database is not updated, there could be no result.

- **which:** This command displays the *complete path of a specified command* by searching the directories assigned to the PATH variable. It's useful in the case when we'd like to find out where a program is installed.
- **whereis:** This command is used to display various details associated with a command. Such as, *location of a command*, *manual pages*, and *sources*. It has several options. For instance, **-b** option searches only for binaries, **-m** option searches only for manual sections, **-s** searches only for sources, **-u** searches for unusual entries.

Hands-on Experience – locate, find

-We can find files or directories by using **locate** and **find** command. Let's take a look at them.



- As we can see, the locate command *didn't return* anything since the *mlocate database was not updated*. On the other hand, let's break down the find command. The **/** represents the location of where Linux will *try to find given item*. The **-type** option provides us to specify the *type what we're looking for, in this case it's directory*. The **-name** option allows us to write down the *name of the item we'd like to find*. Lastly, this command has *returned* the **Absolute Path** of the directory we've been looking for.

-We can also find items by their size.

```
┌──(root㉿kali)-[/home/whiterose]
└─# find /etc -type f -size +300k
/etc/netsniff-ng/oui.conf
/etc/ssh/moduli
```

- In this case, this command searched for files *larger than* **300 KB** under **/etc** directory.

-We can also find items that were modified or created within a certain time.

```
┌──(root㉿kali)-[/home/whiterose]
└─# find /home/whiterose/Desktop/ -type f -mmin -30
/home/whiterose/Desktop/CompTIA/directory/classified.txt
/home/whiterose/Desktop/CompTIA/Linux+
```

- In this case, this command searched for *files modified* within the **last 30 minutes** under **/home/whiterose/Desktop** directory.

-We can also do some complex searching. Let's look at it.

```
┌──(root㉿kali)-[/home/whiterose]
└─# find /etc -type f -size 0 -or -size +30k -mmin -10
/etc/sv/tor/.meta/installed
/etc/sv/ssh/.meta/installed
/etc/sv/dnsmasq/.meta/installed
/etc/chkrootkit/chkrootkit.ignore
/etc/.pwd.lock
/etc/subgid-
/etc/subuid-
/etc/sensors.d/.placeholder
/etc/dictionaries-common/ispell-default
/etc/odbc.ini
/etc/odbcinst.ini
/etc/resolvconf/resolv.conf.d/base
/etc/resolvconf/resolv.conf.d/tail
/etc/security/opasswd
/etc/.java/.systemPrefs/.systemRootModFile
/etc/.java/.systemPrefs/.system.lock
/etc/apt/sources.list~
/etc/apparmor.d/local/usr.sbin.haveged
/etc/apparmor.d/local/usr.bin.tcpdump
/etc/apparmor.d/local/system_tor
/etc/apparmor.d/local/sbin.dhclient
/etc/apparmor.d/local/usr.bin.evince
/etc/apparmor.d/local/torbrowser.Tor.tor
/etc/apparmor.d/local/nvidia_modprobe
/etc/apparmor.d/local/usr.bin.man
/etc/apparmor.d/local/usr.sbin.privoxy
/etc/apparmor.d/local/torbrowser.Browser.firefox
/etc/apparmor.d/local/usr.bin.freshclam
/etc/apparmor.d/local/lsb_release
```

- In this case, this command searched files that have size **between 0 to 30 KB** and modified within the **last 10 minutes** under **/etc** directory.

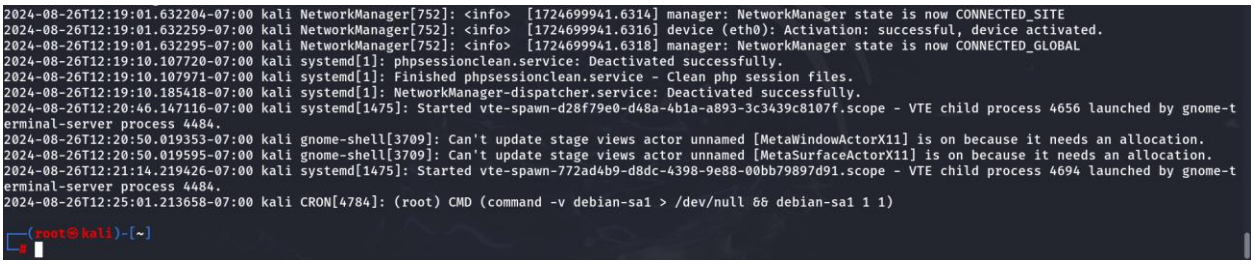
Performing File/Directory Operations

- **cat:** The cat command is short for *concatenate*. It can *display*, *combine*, and *create text files*. Commonly used in *viewing the content of small text files*. It has several options. Such as, **-n** option precedes the output with its specified line number, **-b** option numbers the lines, excluding the blank lines, **-s** option suppresses output of repeated empty lines, **-v** option display non-printing characters as visible characters, **-e** option prints a $ character at the end of each line.
- **head:** This command displays the *first 10 lines of each file by default*. Most common option of this command is **-n** option, which *shows specified number of lines*.
- **tail:** This command displays the *last 10 lines of each file by default*. tail command also has **-n** option that functions as same as head command's. Additionally, **-f** option dynamically watches a file.
- **less & more:** These commands enable users to *display the contents of a file* and *a page through the contents if extended beyond the screen*. Typically, **less** command is commonly used due to its features.
- **cp:** This command *copies* and then *pastes a file* or *directory*. When copying directory, we should use **-R** option. This will copy the directory recursively with everything underneath it going as well. Additionally, to copy the file, user needs to have **execute permissions** on the *source directory* and **write** and **execute permissions** on the *target directory*.
- **mv:** This command *moves files* and *directories* to other locations. It's also used in renaming a file/directory.
- **touch:** This command *tests the permissions* or *creates files*.
- **rm:** This command *removes files/directories*. The **-R** option recursively removes files, subdirectories and the parent directory.
- **unlink:** This command *removes files* but not directories.
- **ls:** This command *lists the contents of directories*. Colors in ls command distinguishes file types. It has several options. Such as, **-l** option displays permission list, owner, group etc., **-F** option displays the nature of the file, **-a** option displays all files, **-R** option recursively displays all subdirectories, **-d** option displays information about symbolic links or directories, **-L** option displays all files in a directory.

- **mkdir:** This command is used to *create a directory*.
- **rmdir:** This command is used to *remove empty directory*. If we want to delete directory with contents, we need to use **rm -R <directory_name>** command.

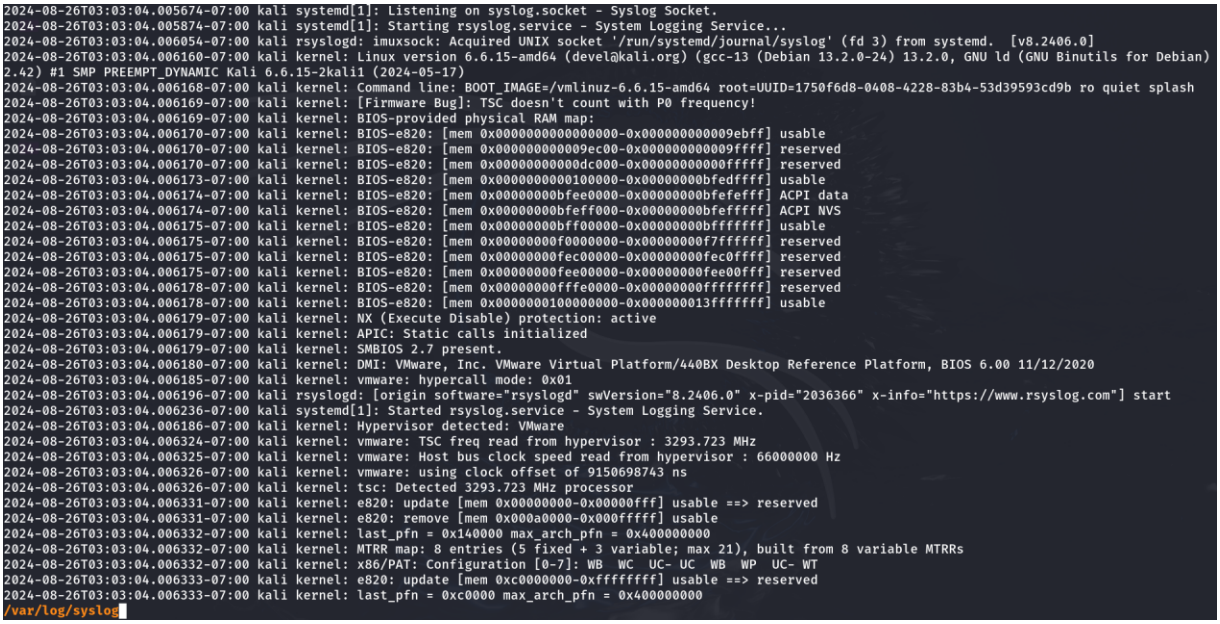
Hands-on Experience – cat, less, head, tail, cp, mv, touch, rm, rmdir

-While viewing the *content of larger files*, the **cat** command *could be useless* since all of the content doesn't fit on the screen.



- In this case, **cat /var/log/syslog** command was executed. As we can see, it shows at the bottom of the file.

-In such cases, we should use **less** command (Viewing the content of larger files).



- In this case, **less /var/log/syslog** command was executed. We can hit space key to go down one line.

-If we want to view the first 10 lines of a file, we can use **head** command.

```
┌──(root㉿kali)-[~]
└─# head /var/log/syslog
2024-08-26T03:03:04.005674-07:00 kali systemd[1]: Listening on syslog.socket - Syslog Socket.
2024-08-26T03:03:04.005874-07:00 kali systemd[1]: Starting rsyslog.service - System Logging Service...
2024-08-26T03:03:04.006054-07:00 kali rsyslogd: imuxsock: Acquired UNIX socket '/run/systemd/journal/syslog' (fd 3) from systemd.  [v8.2406.0]
2024-08-26T03:03:04.006160-07:00 kali kernel: Linux version 6.6.15-amd64 (devel@kali.org) (gcc-13 (Debian 13.2.0-24) 13.2.0, GNU ld (GNU Binutils for Debian)
2.42) #1 SMP PREEMPT_DYNAMIC Kali 6.6.15-2kali1 (2024-05-17)
2024-08-26T03:03:04.006168-07:00 kali kernel: Command line: BOOT_IMAGE=/vmlinuz-6.6.15-amd64 root=UUID=1750f6d8-0408-4228-83b4-53d39593cd9b ro quiet splash
2024-08-26T03:03:04.006169-07:00 kali kernel: [Firmware Bug]: TSC doesn't count with P0 frequency!
2024-08-26T03:03:04.006169-07:00 kali kernel: BIOS-provided physical RAM map:
2024-08-26T03:03:04.006170-07:00 kali kernel: BIOS-e820: [mem 0x0000000000000000-0x000000000009ebff] usable
2024-08-26T03:03:04.006170-07:00 kali kernel: BIOS-e820: [mem 0x000000000009ec00-0x000000000009ffff] reserved
2024-08-26T03:03:04.006170-07:00 kali kernel: BIOS-e820: [mem 0x00000000000dc000-0x00000000000fffff] reserved
```

-Let's say we'd like to see the first two lines of a file. For such cases, we need to use **head** command with **-n** parameter.

```
┌──(root㉿kali)-[~]
└─# head -n 2 /var/log/syslog
2024-08-26T03:03:04.005674-07:00 kali systemd[1]: Listening on syslog.socket - Syslog Socket.
2024-08-26T03:03:04.005874-07:00 kali systemd[1]: Starting rsyslog.service - System Logging Service...
```

-If we want to see the last 10 lines of a file, we can use **tail** command.

```
┌──(root㉿kali)-[~]
└─# tail /var/log/syslog
2024-08-26T12:39:00.727078-07:00 kali systemd[1]: Finished phpsessionclean.service - Clean php session files.
2024-08-26T12:39:01.257719-07:00 kali CRON[4913]: (root) CMD (   [ -x /usr/lib/php/sessionclean ] && if [ ! -d /run/systemd/system ]; then /usr/lib/php/session
clean; fi)
2024-08-26T12:45:01.270853-07:00 kali CRON[4918]: (root) CMD (command -v debian-sa1 > /dev/null && debian-sa1 1 1)
2024-08-26T12:45:46.059721-07:00 kali systemd[1]: Starting chkrootkit.service - chkrootkit...
2024-08-26T12:45:46.471428-07:00 kali crontab[4995]: (root) LIST (nobody)
2024-08-26T12:46:16.209292-07:00 kali kernel: perf: interrupt took too long (2834 > 2500), lowering kernel.perf_event_max_sample_rate to 70500
2024-08-26T12:46:31.238501-07:00 kali chkrootkit-daily[4920]: sending alert to root: [chkrootkit] alert for kali
2024-08-26T12:46:31.963869-07:00 kali systemd[1]: chkrootkit.service: Deactivated successfully.
2024-08-26T12:46:31.964184-07:00 kali systemd[1]: Finished chkrootkit.service - chkrootkit.
2024-08-26T12:46:31.964445-07:00 kali systemd[1]: chkrootkit.service: Consumed 21.374s CPU time.
```

- Let's say we'd like to see the last two lines of a file. For such cases, we need to use **tail** command with **-n** parameter.

```
┌──(root㉿kali)-[~]
└─# tail -n 2 /var/log/syslog
2024-08-26T12:49:00.176903-07:00 kali dbus-daemon[744]: [system] Successfully activated service 'org.freedesktop.nm_dispatcher'
2024-08-26T12:49:00.177141-07:00 kali systemd[1]: Started NetworkManager-dispatcher.service - Network Manager Script Dispatcher Service.
```

-We can copy the files under a directory to another directory by using **cp** command and **-r** option.

```
┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# cp -r /home/whiterose/Downloads/CloudFail /home/whiterose/Tools/

┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# ls -la /home/whiterose/Tools/CloudFail
total 92
drwxr-xr-x 4 root root  4096 Aug 26 13:03 .
drwxr-xr-x 8 root root  4096 Aug 26 13:03 ..
drwxr-xr-x 8 root root  4096 Aug 26 13:03 .git
-rw-r--r-- 1 root root    42 Aug 26 13:03 .gitignore
-rw-r--r-- 1 root root  4482 Aug 26 13:03 DNSDumpsterAPI.py
-rw-r--r-- 1 root root   267 Aug 26 13:03 Dockerfile
-rw-r--r-- 1 root root  1063 Aug 26 13:03 LICENSE.md
-rw-r--r-- 1 root root  2047 Aug 26 13:03 README.md
-rw-r--r-- 1 root root 11453 Aug 26 13:03 cloudfail.py
drwxr-xr-x 2 root root  4096 Aug 26 13:03 data
-rw-r--r-- 1 root root   165 Aug 26 13:03 requirements.txt
-rw-r--r-- 1 root root 30022 Aug 26 13:03 socks.py
-rw-r--r-- 1 root root  2913 Aug 26 13:03 sockshandler.py
```

-We can move files/directories by using **mv** command. Keep in mind that when we move files/directories, they are no longer available at their previous location.

```
┌──(root㉿kali)-[/home/whiterose/Desktop/CompTIA]
└─# mv directory/classified.txt ../

┌──(root㉿kali)-[/home/whiterose/Desktop/CompTIA]
└─# mv /home/whiterose/Desktop/Linux+ .

┌──(root㉿kali)-[/home/whiterose/Desktop/CompTIA]
└─# ls -la /home/whiterose/Desktop/CompTIA
total 16
drwxr-xr-x 3 root      root      4096 Aug 26 13:07 .
drwxr-xr-x 4 whiterose whiterose 4096 Aug 26 13:07 ..
-rw-r--r-- 1 root      root       124 Aug 26 00:34 Linux+
drwxr-xr-x 2 root      root      4096 Aug 26 13:07 directory

┌──(root㉿kali)-[/home/whiterose/Desktop/CompTIA]
└─# ls -ls -la /home/whiterose/Desktop/ | grep clas
-rw-r--r--  1 root      root         67 Aug 25 23:43 classified.txt
```

- In this case, *classified.txt* moved to the Desktop from CompTIA folder, *Linux+* file moved to the CompTIA folder from Desktop.

-We can create files by using **touch** command.

```
┌──(root㉿kali)-[/home/whiterose/Desktop/CompTIA]
└─# touch domain_1.txt

┌──(root㉿kali)-[/home/whiterose/Desktop/CompTIA]
└─# touch domain_2.txt

┌──(root㉿kali)-[/home/whiterose/Desktop/CompTIA]
└─# touch domain_3.txt

┌──(root㉿kali)-[/home/whiterose/Desktop/CompTIA]
└─# ls -la
total 16
drwxr-xr-x 3 root      root      4096 Aug 26 13:11 .
drwxr-xr-x 4 whiterose whiterose 4096 Aug 26 13:07 ..
-rw-r--r-- 1 root      root       124 Aug 26 00:34 Linux+
drwxr-xr-x 2 root      root      4096 Aug 26 13:07 directory
-rw-r--r-- 1 root      root         0 Aug 26 13:11 domain_1.txt
-rw-r--r-- 1 root      root         0 Aug 26 13:11 domain_2.txt
-rw-r--r-- 1 root      root         0 Aug 26 13:11 domain_3.txt
```

-If we want to remove a directory which includes subdirectories/files, we *CANNOT delete* it by using **rmdir**. Instead, we need to use **rm** command with **-r** option.

```
┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# rmdir CompTIA
rmdir: failed to remove 'CompTIA': Directory not empty

┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# rm -r CompTIA

┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# ls -la | grep CompTIA
```

- As we can see, we can only use rmdir when we want to remove an empty directory.

Processing Text Files

- **echo:** This command *prints out arguments as standard output*. This command is commonly used to display text strings or command results as messages to the screen.
- **printf:** This command provides the user with more control over how the output is formatted.

- **tr:** This command performs operations like *removing repeated characters*, *converting uppercase to lowercase*, and *basic character replacement* and *removal*.
- **wc:** This command allows users to *count the number of lines*, *words*, *characters* and *bytes in a file* and prints the result. It has several options. Such as, **-c** option to display byte count, **-m** option to display character count, **-l** option to display the newline count, **-w** option to display the word count.
- **sort:** This command is used for *sorting lines of text files*. It has several options. Such as, **-k** option to specify field values, **-n** option to compare and sort lines based on the string numerical value, **-r** option to sort fields in descending order, **-t** option to separate one field from another.
- **cut:** This command *extracts the specified lines of text from a file*. It has several options. Such as, **-c** option specifies the number of the character to cut from each line, **-d** option separates one field from another, **-f** option specifies the field numbers to cut as separated by the delimiter, **-s** option suppresses a line if the delimiter is not found.
- **paste:** This command is used to *merge lines from text files horizontally*. It uses tab space as a delimiter by default.
- **diff:** This command *compares text files*. On the output, symbols like < and > indicate that line should be removed and added. It has several options. For instance, **-b** option ignores spacing differences, **-i** option ignores case differences, **-t** option expands tab characters in output lines, **-w** option ignores spacing differences and tabs, **-c** option displays a list of differences with three lines of context, **-u** option shows results in unified mode.
- **grep:** This command *searches file contents for specific strings or patterns*. It has several options. Such as, **-E** option matches a pattern as an extended regular expression, **-F** option matches a pattern as a list of fixed strings, **-f** option matches patterns contained in a specific file, **-i** option ignores casing, **-v** option displays only the lines that don't match provided pattern, **-c** option prints only the number of matching lines. We can also use grep to search for an item inside of a directory.

- **awk:** This command *performs pattern matching on files* and also *text processing*. Patterns and actions are specified within single quotes. It can be used to extract, delete, and modify text based on patterns.
- **sed:** It's a *stream editor for text file modification*. It can delete lines, substitute text and more. This command can also be used for global search and replace actions. It has several options. Such as, **-d** option to delete the lines that match a specific pattern, **-n** or **-p** options to print only the lines that contain the pattern.
- **ln:** This command *creates links to files*. The link does not contain any data, only a reference to the target file. There are two types of links. **Hard Link** and **Symbolic Link**. Hard Link is a reference to another file. If the original item is deleted after a hard link, the contents will still be available in the linked file. Symbolic (Soft) Link reference to an item that can span multiple file systems. If the original item is deleted, the contents will be lost.

Hands-on Experience – cat, wc, diff, grep, ln

-We can see the contents of a file with number of lines by using **cat** command with **-n** parameter.

```
┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# cat -n certifications.txt
     1  CompTIA Linux+:
     2  Domination in Linux
     3  Linux System Admin
     4
     5  CompTIA Security+:
     6  Basic Security Test
     7
     8  OSCP:
     9  Based on Offensive Security
    10  Penetration Tester
    11
```

-We can see the number of lines, words, and characters by using **wc** command.

```
┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# wc certifications.txt
11  20 151 certifications.txt
```

- First value represents number of lines, second value represents number of words, and third value represents number of characters.

-We can compare two files by using **diff** command.

```
┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# diff tools_1.txt tools_2.txt
1,4c1,4
< Metasploit
< Nmap
< Ghidra
< cutter
---
> BurpSuite
> Wireshark
> Maltego
> beef
6,7c6,7
< wp-scan
< searchsploit
---
> mimikatz
> sqlmap
9c9
< linpeas
---
> nikto
10a11,12
> crunch
> radare2
```

- Arrows pointed to right indicate what lines need to be added to the first file to replicate second file.
- In this case, 1, 4c, 4 tells us to replace the words in the lines from 1 to 4 with the words indicated by arrows pointed to right. Then, replace the lines from 6 to 7, as well as replace 9th line and 11th and 12th line. As a hint, we can think of words written in green are going to be replaced with the words written in red.

-Let's say we're dealing with a large file, and we'd like to see a specific part. In such cases, we can use **grep** command.

```
┌──(root㉿kali)-[/home/whiterose]
└─# grep "apt install" /root/.zsh_history
apt install git curl clang gcc gdb python3 tor torbrowser-launcher -y
apt install resolvconf
apt install dnsmasq arm privoxy -y
apt install dnsmasq privoxy -y
apt install wifiphisher -y
apt install autopsy -y
apt install crunchy -y
apt install crunch -y
apt install nikto -y
apt install burpsuite -y
apt install hashcat -y
apt install beef-xss -y
apt install nuclei -y
apt install burp -y
apt install dirb feroxbuster -y
apt install wpscan -y
apt install gobuster -y
```

- In this case, "*apt install*" keywords have been searched in **/root/.zsh_history** file.

-We can use **ln** command to link files/directories. By default, ln command performs *hard link*.

```
┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# ln /home/whiterose/Desktop/fsociety.jpg /home/whiterose/mr_robot
```

- In this case, this command created a hard link named '*mr_robot*' for the file '*fsociety.jpg*'.

-Keep in mind that for hard link, the file systems have to be the *same*. Let's try it.

```
┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# ln /Additional/Tools/macchanger.py /home/whiterose/Desktop/macchanger_link
ln: failed to create hard link '/home/whiterose/Desktop/macchanger_link' => '/Additional/Tools/macchanger.py': Invalid cross-device link

┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# lsblk -f
NAME                FSTYPE      FSVER            LABEL                UUID                                   FSAVAIL FSUSE% MOUNTPOINTS
sda
├─sda1              ext4        1.0                                   0e6e6aaf-333e-44a6-a9a7-dfd652843430    668.8M    20% /boot
├─sda2              swap        1                                     6bab8a0e-e7db-47ce-a5ec-ce5b6f861822                  [SWAP]
├─sda3              ext4        1.0                                   1750f6d8-0408-4228-83b4-53d39593cd9b     11.3G    62% /
└─sda4              ext4        1.0                                   8e595d1c-6981-4b98-8b80-2407523f5b17     19.2G     5% /home
sdb
├─sdb1              LVM2_member LVM2 001                              vPn2XV-XGv8-rHLV-wy4U-Ift0-vGLP-Ya6udc
│ └─extra-additional ext4       1.0                                   21610b40-6a7b-4820-854c-9da9f84444da      1.8G     0% /Additional/PDFs
└─sdb2              LVM2_member LVM2 001                              kISYla-0Gce-DKfF-sCBf-Wtqn-P2Ec-8LtTRj
  ├─extra-additional ext4       1.0                                   21610b40-6a7b-4820-854c-9da9f84444da      1.8G     0% /Additional/PDFs
  └─extra-tools     xfs                                               52472519-30e1-42f1-9c01-489500254d94    909.3M     5% /Additional/Tools
sr0                 iso9660     Joliet Extension Kali Linux amd64 1 2024-05-27-17-38-40-00
```

- As we can see, it returned **Invalid cross-device link**. */home/* directory has ext4 as file system while */Additional/Tools* has XFS file system.

-We can perform linking from different file systems by using soft linking. To do that, we use **ln** command with **-s** parameter.

```
┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# ln -s /Additional/Tools/macchanger.py /home/whiterose/Desktop/macchanger_link
```

- Soft links reference in the existing file. Hard links create a copy of the existing file and constantly updates the file.

Manipulating File Output

- **Text Streams:** Text streams are *sequence of lines of text* that can be used for reading from or writing to system components and devices, such as files, Command Line Interface, Network Sockets. There are typically three types of streams:
    1. **Standard Input:** Source for command input. Also known as **stdin**.

2. **Standard Output:** Destination for command output. Also known as **stdout**.
3. **Standard Error:** Used as the destination for error messages. Also known as **stderr**.

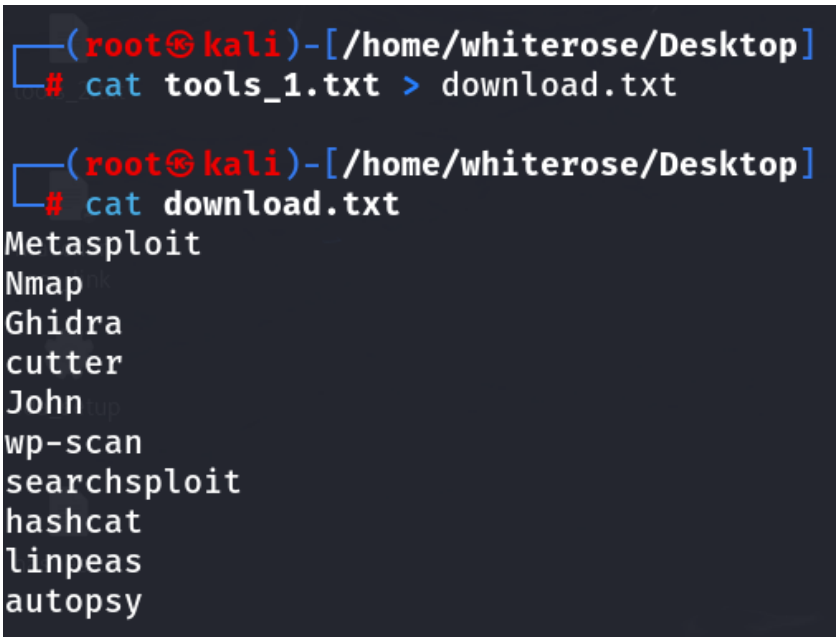
-It's possible to redirect the input, output and error of various commands to and from different files. Let's take a look at redirection operators.

- **>:** Redirects standard output to a file.
- **>>:** Appends standard output to the end of the destination file.
- **2>:** Redirects standard error messages to a file.
- **2>>:** Appends standard error messages to the end of the destination file.
- **&>:** Redirects both standard output and error to a file.
- **<:** Reads input from a file instead of the keyboard.
- **<<:** Provides input data until a line containing the special string.
- **|:** Pipes are used to combine standard input/output streams of commands.

- **Piping:** Involves *combining the standard output of one command as the standard input for another command*. To perform this operation, **|** (pipe) operator is used.
- **xargs:** This command *reads streams of data from standard input*, then *generates* and *executes command lines*. It'd be useful when processing multiple arguments from standard input.
- **tee:** This command *reads the standard input, sends the output to the default output device*, and *copies the output to each specified file*. Useful when we want to write the output of a command into a file.
- **Terminal Redirection:** In Linux, terminals can be identified in the format **/dev/tty<number>**. Standard input and output can be redirected to different running processes by referencing **/dev/tty<number>**.

Hands-on Experience – Redirection, Piping, tr, sort, tee

-We can *write* the output of a command inside a file by using > operator. This operation performs that getting the command's output as STDIN and redirect it inside the specified file.



- In this case, the *download.txt* file is being created, and contents of *tools_1.txt* file have been written into it.

-Difference between **single** > operator and **double** > operator (>>) is that the **single** > **operator** write only the redirected command's output to file. So, if the provided file contains data, it will be replaced by the output of redirected command. On the other hand, double > operator (>>) appends the redirected command's output to file. So, it does not replace any data if the file contains data.

-We can *append* the output of a command by using >> operator.

```
┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# cat tools_2.txt >> download.txt

┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# cat download.txt
Metasploit
Nmap
Ghidra
cutter
John
wp-scan
searchsploit
hashcat
linpeas
autopsy

BurpSuite
Wireshark
Maltego
beef
John
mimikatz
sqlmap
hashcat
nikto
autopsy
crunch
radare2
```

- In this case, contents of *tools_2.txt* file have been added at the bottom of the *download.txt* file.

-We can change the characters in the file to uppercase or lowercase by using **tr** command.

```
┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# cat -n manipulating_file_output.txt
     1  kali linux
     2  parrot os
     3  ubuntu
     4  linux mint

┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# tr l L < manipulating_file_output.txt
kaLi Linux
parrot os
ubuntu
Linux mint
```

- In this case, lowercase 'L's changed to uppercase 'L'

-Let's get things a little complicated. Assume that we have an unsorted user list, we need to sort it excluding first line, and we need to write it to a new file. To do this, we will be using both **piping** (|) and **redirection operator** (>).

```
┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# cat user_list.txt
Jakob
Darlene
Marty
Heisenberg
Michael
Lincoln
Wendy
Alex
Sarah

┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# tail -n +2 user_list.txt | sort -k1 > user_order.txt

┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# cat user_order.txt
Alex
Darlene
Heisenberg
Lincoln
Marty
Michael
Sarah
Wendy
```

- In this case, first line is excluded by using **tail -n +2** and the file is sorted by **sort -k1** command. The piping operator *passes the first command's output as input to second command*. Lastly, output is redirected to a new file called *user_order.txt*.

-If we want to both redirect the output to file and also print it on the screen, we can use **tee** command.

```
┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# sort -k1 manipulating_file_output.txt | tee os.txt
kali linux
linux mint
parrot os
ubuntu
```

- In this case, *manipulating_file_output.txt* file has been sorted and redirected to *os.txt* as well as result printed on the screen.

-When we're searching for files/directories in the system by using **find** command, sometimes irrelevant items could be displayed because of **permission denied** error. In order to prevent this, we can use **2>/dev/null** to redirect standard errors to the /dev/null location which is a black hole in Linux systems.

```
┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# find /home/whiterose/ -type f -name 'os.*' 2>/dev/null
/home/whiterose/Desktop/os.txt
```

Kernel Modules

Linux Kernel

-Kernel is core of the system. Kernel responsible for **System Initialization**, **Process Scheduling**, and **Memory & Hardware Management**. Kernel *manages File System Access*, *Memory*, *Processes*, *Devices*, and *other resource allocation* on the system. Kernel divides software that's running in memory into two spaces:

1. **Kernel Space:** Kernel space provides *abstraction for security*, *hardware*, and *internal data structures*. It is where the kernel executes services.
2. **User Space:** User space refers to all of the code in an operating system that *lives outside of the kernel*. For example, programs, tools, programming languages are *user space applications*. User space applications get access to data by making special requests to kernel called **system calls**.

-Kernels can be classified as monolithic or microkernel.

- **Monolithic Kernel:** It is responsible for *running all system modules*, such as device drivers or file systems in kernel space. It can interact quickly with all different devices, but it *consumes more memory*.

- **Microkernel:** It runs the *minimum amount of resources*, has larger user space. Microkernel offers *better stability and security* however it may have *lower performance* compared to monolithic kernel.

- **Device Driver:** Device Drivers are software programs that enable the *operating system to communicate with hardware devices*, such as printers, keyboard, mice etc. Device Drivers can be included in the operating system, or we can install them afterwards.
- **Linux Kernel:** Linux Kernel is a free and open-source *monolithic kernel* that manages all other resources and hardware device on an operating system. It offers features like virtual memory management, TCP/IP Networking support, shared libraries etc. Linux kernel also offer modularity which enables users to configure and extend kernel functionality to meet their specific needs.

-Linux Kernel is continuously updated and given version numbers for identification. Format of the version number is major on the left-hand side and minor on the right-hand side, and it changes periodically based on major developments.

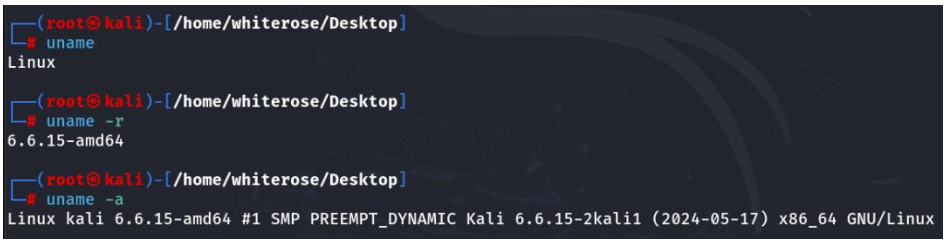
- **uname:** This command is used to *print name of the kernel*. We can also view kernel version by using **-r** parameter.

-Kernel operates in several layers in kernel space.

- **System Call Interface (SCI):** Handles system calls sent from user applications to the kernel.
- **Process Management:** Handles different processes by allocating separate execution space on the processor.
- **Memory Management:** Manages the computer's memory.
- **File System Management:** Manages the filesystem.
- **Device Management:** Manages devices by controlling device access and interfacing between user applications and hardware devices on the computer.

Hands-on Experience – uname

-We can display the kernel information by using **uname** command.



- The **-r** parameter displays kernel version, **-a** parameter displays all information.

Linux Kernel Modules

-Kernel modules extend the functionality of the Linux kernel. Kernel modules have the **.ko** file extension, and they're specific to particular kernel versions.

- **/usr/lib/modules/ directory:** This directory contains the *modules of different kernel versions*. Modules are stored in different subdirectories based on their categories. These categories are:
  - ***arch:*** Architecture-specific support.
  - ***crpyto:*** Encryption and other cryptographic functions.
  - ***drivers:*** Hardware components.
  - ***fs:*** File systems.
  - ***net:*** Networking components.
- **lsmod:** This command *displays the currently loaded kernel modules*.
- **modinfo:** This command *displays information about a particular kernel module*.
- **insmod:** This command *installs a module into the currently running kernel*.
- **rmmod:** This command *removes a module from the currently running kernel*.
- **modprobe:** Used to *load* or *unload modules from a kernel*. The *modprobe* command can be used instead of **insmod** and **rmmod** commands. To add modules, we can use **-a** option. To remove modules, we can use **-r** option.
- **depmod:** This command *updates database of dependencies*, allowing modprobe command to load modules and their dependencies accurately.

-Configuration settings for kernel modules are stored in files with .conf file extension underneath **/etc/modprobe.d/** directory. We can define module aliases, blacklist modules, or specify commands to run when loading a particular module.

-Users can change some kernel parameter at runtime, such as *hardening security*, *configuring network limitations*, *virtual memory settings* etc. In order to do this, we will be using **/proc/sys/** directory.

- **/proc/sys/ directory:** Lists the parameters to configure on a system. /proc/sys/ directory is divided into several categories, including *crypto*, *debug*, *dev*, *fs* (file system), *kernel*, *net* (networking), *user*, *vm* (virtual memory).
- **sysctl:** This command is used to *view* or *set kernel parameters at runtime*. So, we don't have to recompile the kernel or reboot the system when modifying kernel parameters. It has several options. Such as, **-a** option to display all parameters and current values, **-w** option to set a parameter value, **-p** option to load sysctl settings from the specified file, **-e** option to ignore errors, **-r** option to apply command to parameters matching a given pattern.
- **/etc/sysctl.conf file:** This file allows for *configuring changes to the running Linux kernel*, including network, security, and logging settings.

Hands-on Experience – lsmod, modinfo, depmod, modprobe

-We can list out the current running kernel modules by using **lsmod** command.

```
┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# lsmod
Module                  Size  Used by
cpuid                  12288  0
nf_tables             372736  0
battery                28672  0
mptcp_diag             12288  0
xsk_diag               12288  0
vsock_diag             12288  0
tcp_diag               12288  0
udp_diag               12288  0
raw_diag               12288  0
inet_diag              28672  4 tcp_diag,mptcp_diag,raw_diag,udp_diag
unix_diag              12288  0
af_packet_diag         12288  0
netlink_diag           12288  0
snd_seq_dummy          12288  0
snd_hrtimer            12288  1
snd_seq_midi           20480  0
snd_seq_midi_event     16384  1 snd_seq_midi
snd_seq               114688  9 snd_seq_midi,snd_seq_midi_event,snd_seq_dummy
qrtr                   57344  4
vsock_loopback         12288  0
vmw_vsock_virtio_transport_common    61440  1 vsock_loopback
vmw_vsock_vmci_transport    45056  1
vsock                  61440  6 vmw_vsock_virtio_transport_common,vsock_loopback,vsock_diag,vmw_vsock_vmci_transport
binfmt_misc            28672  1
snd_ens1371            36864  1
btusb                  86016  0
snd_ac97_codec        196608  1 snd_ens1371
btrtl                  32768  1 btusb
btintel                57344  1 btusb
ac97_bus               12288  1 snd_ac97_codec
btbcm                  24576  1 btusb
gameport               28672  1 snd_ens1371
btmtk                  16384  1 btusb
snd_rawmidi            53248  2 snd_seq_midi,snd_ens1371
intel_rapl_msr         20480  0
bluetooth            1134592  6 btrtl,btmtk,btintel,btbcm,btusb
vmw_balloon            32768  0
snd_seq_device         16384  3 snd_seq,snd_seq_midi,snd_rawmidi
intel_rapl_common      36864  1 intel_rapl_msr
```

-We can display list of information about a specified kernel module by using **modinfo** command.

```
┌──(root㉿kali)-[/lib/…/6.6.15-amd64/kernel/drivers/bluetooth]
└─# modinfo btusb.ko.xz
filename:       /lib/modules/6.6.15-amd64/kernel/drivers/bluetooth/btusb.ko.xz
license:        GPL
version:        0.8
description:    Generic Bluetooth USB driver ver 0.8
author:         Marcel Holtmann <marcel@holtmann.org>
```

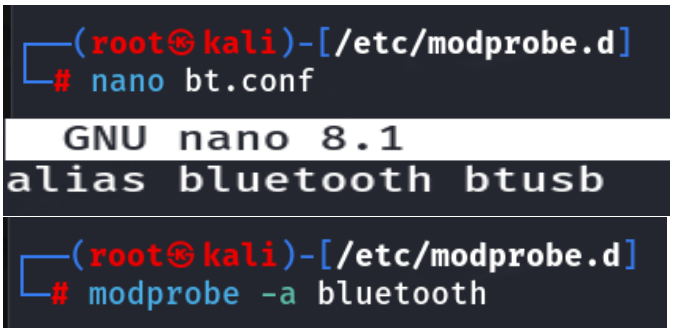
- In this case, Bluetooth module has been examined.

-We can update dependencies by using **depmod** command.

```
┌──(root㉿kali)-[/lib/…/6.6.15-amd64/kernel/drivers/bluetooth]
└─# depmod
```

- It might take a while to update dependencies.

-Let's configure an alias for a module. We need to create aliases within the **/etc/modprobe.d** directory.



- In this case, alias name is *bluetooth* for the **btusb** (actual drive) module. Then, to add this alias to modules, **modprobe -a bluetooth** (alias name) command was used.

Monitoring Kernel Modules

- **/proc/ directory:** /proc/ directory is a *Virtual File System* (VFS) which *provides information about the kernel's running processes*. Let's see the key ones in detail.
  - ***/proc/cmdline:*** Contains options passed to the kernel by the *boot loader*.
  - ***/proc/cpuinfo:*** Contains *CPU information*.
  - ***/proc/devices:*** Contains a list of *character* and *block device drivers* loaded into the currently running kernel.
  - ***/proc/filesystems:*** Contains a list of *file systems types* that are supported by kernel.
  - ***/proc/meminfo:*** Contains information about *RAM usage*.
  - ***/proc/modules:*** Contains information about *modules currently installed* on the system.
  - ***/proc/stat:*** Contains *various statistics* about the system's last reboot.
  - ***/proc/version:*** Contains several points of information about the Linux kernel. These are *version of the GNU Compiler Collection* (GCC) used for compilation, *compilation time*, and the *compiler's username*.
- **GNU Compiler Collection (GCC):** Used to *compile the kernel*, the *username*, and the *time the kernel was compiled*.
- **dmesg:** This command is used to *print messages that have been sent to the kernel's message during* and *after system boot*, **commonly used for troubleshooting system issues**. Device drivers and other kernel components send messages to the buffer, including diagnostic messages in case of errors.

The dmesg command stands for *display message* or *driver message*. It has several options. Such as, **-c** option to clear the kernel buffer after printing, **-f** option to restrict output by facility, **-l** option to restrict output by message level, **-H** option to display in human-readable format.

Hands-on Experience – dmesg

-To get information about the kernel, we can read **/proc/version** file.

```
┌──(root㉿kali)-[/etc/modprobe.d]
└─# cat /proc/version
Linux version 6.6.15-amd64 (devel@kali.org) (gcc-13 (Debian 13.2.0-24) 13.2.0, GNU ld
(GNU Binutils for Debian) 2.42) #1 SMP PREEMPT_DYNAMIC Kali 6.6.15-2kali1 (2024-05-17)
```

-We can display and examine kernel message buffer by using **dmesg** command.

```
┌──(root㉿kali)-[/etc/modprobe.d]
└─# dmesg -H
[Aug27 22:22] Linux version 6.6.15-amd64 (devel@kali.org) (gcc-13 (Debian 13.2.0-24) >
[  +0.000000] Command line: BOOT_IMAGE=/vmlinuz-6.6.15-amd64 root=UUID=1750f6d8-0408->
[  +0.000000] [Firmware Bug]: TSC doesn't count with P0 frequency!
[  +0.000000] BIOS-provided physical RAM map:
[  +0.000000] BIOS-e820: [mem 0x0000000000000000-0x000000000009ebff] usable
[  +0.000000] BIOS-e820: [mem 0x000000000009ec00-0x000000000009ffff] reserved
[  +0.000000] BIOS-e820: [mem 0x00000000000dc000-0x00000000000fffff] reserved
[  +0.000000] BIOS-e820: [mem 0x0000000000100000-0x00000000bfedffff] usable
[  +0.000000] BIOS-e820: [mem 0x00000000bfee0000-0x00000000bfefefff] ACPI data
[  +0.000000] BIOS-e820: [mem 0x00000000bfeff000-0x00000000bfefffff] ACPI NVS
[  +0.000000] BIOS-e820: [mem 0x00000000bff00000-0x00000000bfffffff] usable
```

-We can only display messages that are giving us warning messages by using **dmesg -l warn** command.

```
┌──(root㉿kali)-[/etc/modprobe.d]
└─# dmesg -H -l warn
[Aug27 22:22] [Firmware Bug]: TSC doesn't count with P0 frequency!
[  +0.000142] unchecked MSR access error: RDMSR from 0x852 at rIP: 0xffffffff8528d467 (nati>
[  +0.000007] Call Trace:
[  +0.000002]  <TASK>
[  +0.000002]  ? ex_handler_msr+0x121/0x130
[  +0.000002]  ? fixup_exception+0x234/0x310
```

-We can only display error message by using **dmesg -l err** command.

```
┌──(root㉿kali)-[/etc/modprobe.d]
└─# dmesg -H -l err
[Aug27 22:22] [Firmware Bug]: cpu 0, try to use APIC520 (LVT offset 2) for vector 0xf4, but>
[  -0.411507] [Firmware Bug]: cpu 1, try to use APIC520 (LVT offset 2) for vector 0xf4, but>
[  +0.000000] [Firmware Bug]: cpu 2, try to use APIC520 (LVT offset 2) for vector 0xf4, but>
```

Linux Boot Process

Linux Boot Components

- **Booting:** Booting is the process of *starting* or *restarting a computer* and *loading an operating system*. In boot process, booting environment reads a small program stored in *Read Only Memory* (ROM) which then executes operations in *Random Access Memory* (RAM) to bootstrap the operating system.
- **Boot Loader:** Boot Loader is a *small program stored in ROM* that loads the kernel from a storage device. Boot loaders protect the boot process with a password to prevent unauthorized booting of the system. Boot loader includes three main components.
    1. **Boot Sector Program:** Loads the *second boot loader* on startup.
    2. **Second Stage Boot Loader:** Loads the operating system and contains a kernel loader.
    3. **Boot Loader Installer:** Controls the installation of drive sectors and runs only when booting.

-There are several boot options where we can boot Linux from BIOS or UEFI.

- **BIOS (Basic Input/Output System):** BIOS is a *firmware interface standard stored on ROM*. BIOS enables to test the various hardware components in a computer as well as run a boot loader.
- **UEFI (Unified Extensible Firmware Interface):** UEFI is a *newer firmware* compared to BIOS. UEFI provides *faster performance*, *larger memory access*, and *more improved security*. Both BIOS and UEFI include the ability to set a password.

-Additionally, system can be booted from various sources. Such as, *ISO images*, *Preboot Execution Environment* (PXE) over the network, *HTTP/FTP for network booting*, and *NFS* (Network File System) for network booting.

- **PXE (Preboot Execution Environment) Boot:** PXE is a set of standards that enables a computer to load an operating system over a network connection, in other words from NIC (Network Interface Card). Once the system is booted, the DHCP server gives an IP address to the system, and

configurations from DHCP server will lead the PXE client to communicate with TFTP (Trivial File Transfer Protocol) server.

- **Master Boot Record (MBR):** Information about *how a hard disk has been partitioned is going to be stored inside MBR*. MBR is a sector that BIOS reads in and starts when system is booted.
- **GUID Partition Table (GPT):** GPT is also a sector which is a *newer version of MBR*. GPT has a partition structure with a more modern design, and it is part of the UEFI standard.
- **Raw Partition:** Raw partition enables users and applications to *read from* and *write to a block storage without using the system cache*.
- **Initial RAM Disk (initrd):** It *acts as temporary root file system* during system boot.
- **initrd image:** It's an *archive file that contains all the essential files* that are required for booting the operating system. Stored underneath the /boot directory.
- **mkinitrd:** This command is used to *create the initrd image* for preloading the kernel modules.
- **/boot/ directory:** This directory contains files related to boot, such as *GRUB configuration files* (**/boot/grub** & **/boot/grub2**), *EFI boot files* (**/boot/efi**), *initrd/initramfs images* (**/boot/initramfs-<kernel_version>.img**), *Linux kernel* (**/boot/vmlinuz-<kernel_version>**).
- **dracut:** This command *generates initramfs images* for Linux systems, replacing mkinitrd in some distributions.

-Boot process in Linux follows sequential steps.

1. The processor checks the BIOS/UEFI firmware.
2. The BIOS/UEFI checks for bootable media (HDD, SSD, USB-drives etc.)
3. The BIOS/UEFI loads the primary boot loader.
4. GRUB 2 selects the operating system.
5. The boot loader determines the kernel and locates the kernel binary.
6. The kernel configures the available hardware drivers.
7. The kernel mounts the main root partition and releases unused memory.
8. The systemd program searches for the default.target file.

9. The system authenticates the user.
10. The system is ready to use.

- **Kernel Panic:** It occurs when a *fatal error is detected*. It can happen during boot due to issues like corrupted kernel, root file system errors, hardware incompatibility, or initrd/initramfs problems.

Configuring GRUB 2

- **GNU Grand Unified Bootloader (GNU GRUB):** Allows users to choose which operating system or kernel version to boot. GNU GRUB is the *primary bootloader* for most modern distributions. However, GRUB 2 became the default bootloader nowadays.

-GRUB is phased out in favor of the new version of GRUB known as **GRUB 2**. GRUB 2 offers more control over the boot process, devices and behavior. Some of the improvements in GRUB 2 are followed by:

- Non-86 architecture platforms,
- Boot OS from storage media,
- Partition UUIDs and loading modules,
- Configure bootloader through scripts,
- Customization features.

- **grub2-install:** This command is used to *install GRUB 2* on BIOS systems. The grub2-install command copies GRUB 2 files into the /boot/grub2/ directory, and even installs GRUB 2 into the boot sector on some platforms.

-Main configuration file for the GRUB 2 is **grub.cfg**, located in **/boot/grub2/** on *BIOS* systems and **/boot/efi/EFI/<distribution_name>/** directory on *UEFI* systems.

- **/etc/grub.d/ directory:** This directory contains scripts used to build the main *grub.cfg* file.
  - ***/etc/grub.d/40_custom file:*** This file allows us to customize the boot menu presented to user.
- **grub2-mkpasswd-pbkdf2:** This command *generates a password hash for protecting the GRUB 2 boot menu*.

- **/etc/default/grub file:** This file *contains GRUB 2 display menu settings* that are read by the scripts underneath /etc/grub.d/ directory, and they are going to be built into our grub.cfg file.
- **grub2-mkconfig:** This command *generates grub.cfg file* or *updates the existing grub.cfg file*. It combines configuration file templates in /etc/grub.d/ directory with settings in /etc/default/grub to create grub.cfg file.

System Components

Configuring Localization Options

- **Localization:** Localization means *adapting system components for use within a distinct culture* or *language*. Localization includes *configuring time zone*, *keyboard layout*, *appropriate date and time formats based on region* etc. For instance, using the *correct time zone* is important for system processes. The **cron daemon**, which is used for scheduling repetitive tasks, uses the system's time zone for executing cron jobs.
- **/usr/share/zoneinfo/ directory:** This directory comprises *all the regional time zones* that system can use. Subdirectories of /usr/share/zoneinfo directory organize time zones by *language* and *region*. In order to change the time zone, we need to create a *symbolic link* from one of these files to */etc/localtime* file.
- **/etc/timezone file:** In some *Debian-based* distributions, */etc/timezone* file is used to specify the time zone of the system.
- **date:** This command *prints the date* in a specified format based on the /etc/locatime file. By default, it's going to print in the following format: **<day of the week> <month> <day><24-hour time><time zone><year>**. We can use formatting options to customize the displayed date format.
- **timedatectl:** This command *sets system date* and *time information*. Subcommands of timedatectl command allow us to *manage date* and *time information*. The timedatectl command is going to expose three different clocks that the system uses. They are *Local Clock*, *Universal Time Clock (UTC)*, and *Hardware Clock (RTC)*.

- **hwclock:** This command allows for *viewing* and *setting of the hardware clock*. It's recommended that keeping the hardware clock aligned with UTC prevents time correction issues.
- **/etc/adjtime file:** This file records information about when and by how much the hardware clock is changed.
- **locatectl:** This command is used to *display* and *configure the system locale* and *keyboard layout settings*. The localectl command includes both subcommands and options.
- **Character Encoding & Decoding:** Character Encoding is the process of *converting text into bytes* whereas Character Decoding is the process of *converting bytes into text*. In most of the systems, default encoding is UTF-8 using the Unicode character set.

Hands-on Experience – timedatectl, localectl

-We can display the time and date information by using **timedatectl** command.

```
┌──(root㉿kali)-[/etc/modprobe.d]
└─# timedatectl
               Local time: Mon 2024-09-02 05:00:58 PDT
           Universal time: Mon 2024-09-02 12:00:58 UTC
                 RTC time: Mon 2024-09-02 12:00:58
                Time zone: America/Los_Angeles (PDT, -0700)
System clock synchronized: yes
              NTP service: active
          RTC in local TZ: no
```

- In this case, the system is using **America/Los Angeles** region as time zone.

-We can display all the available time zones in the system, in other words time zones under /usr/share/zoneinfo, by using **timedatectl** command with **list-timezones** subcommand.

```
┌──(root㉿kali)-[/etc/modprobe.d]
└─# timedatectl list-timezones
Africa/Abidjan
Africa/Accra
Africa/Addis_Ababa
Africa/Algiers
Africa/Asmara
Africa/Asmera
Africa/Bamako
Africa/Bangui
Africa/Banjul
Africa/Bissau
Africa/Blantyre
Africa/Brazzaville
Africa/Bujumbura
Africa/Cairo
Africa/Casablanca
Africa/Ceuta
Africa/Conakry
Africa/Dakar
Africa/Dar_es_Salaam
```

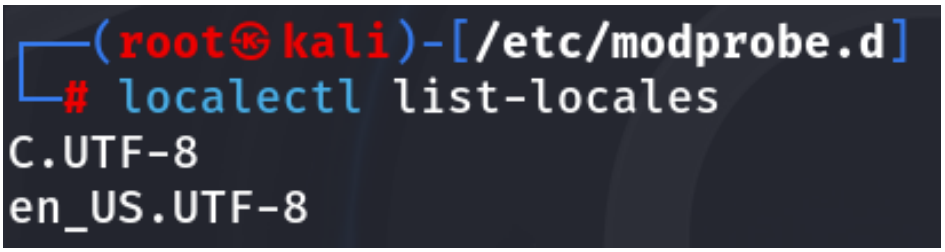
-We can change the time zone by using **timedatectl** command with **set-timezone** subcommand.

```
┌──(root㉿kali)-[/etc/modprobe.d]
└─# timedatectl set-timezone Europe/Amsterdam


┌──(root㉿kali)-[/etc/modprobe.d]
└─# timedatectl status
               Local time: Mon 2024-09-02 14:49:33 CEST
           Universal time: Mon 2024-09-02 12:49:33 UTC
                 RTC time: Mon 2024-09-02 12:49:34
                Time zone: Europe/Amsterdam (CEST, +0200)
System clock synchronized: yes
              NTP service: active
          RTC in local TZ: no
```

- In this case, time zone has switched to **Europe/Amsterdam**.

-We can display system's current language and keyboard layout by using **localectl** command.

-We can list locales by using **localectl** command with **list-locales** subcommand.



-We can set locale by using **localectl** command with **set-locale** subcommand.

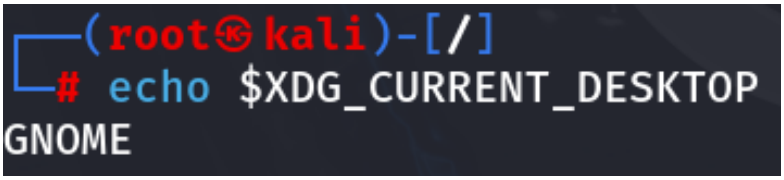
Configuring Graphical User Interface (GUI)

-Configuring Graphical User Interface (GUI) in Linux involves selecting and customizing the GUI environment to meet the needs and preferences of users. GUIs provide a visual interface for interacting with the system and are a common choice for workstations and personal desktops.

- **Graphical User Interface (GUI):** A GUI is *visualized interface* which provides interactions with system or application and allows users to perform tasks by using graphical elements such as menus, windows.
- **Display Server:** A Display Server is the component of GUI that constructs and manages the windowing system and other visual elements that can be drawn on the screen.
- **Desktop Environment:** Linux offers various desktop environments that we can choose from, each of them has its own style and features. Some of the popular desktop environments include *GNOME*, *KDE Plasma*, *XFCE*, *Cinnamon*, *MATE* etc.
- **Remote Desktop:** Remote Desktop is a concept that *allows users to access a remote system's desktop environment over a network*. One of the most popular remote desktop services is called Virtual Network Computing (VNC). VNC is a cross-platform remote desktop service that enables full remote control of a desktop environment.
- **Secure Shell (SSH):** Secure Shell is a *remote access protocol* that *encrypts transmissions over a network*.
- **SSH Port Forwarding:** SSH Port Forwarding is the process of *tunneling an application through SSH* to secure it in the transmission. SSH Port Forwarding allows us to establish an encrypted connection for services.
- **Console Redirection:** Console Redirection allows us to remotely control and manage systems, even at the BIOS or UEFI level, by forwarding input and output through a serial connection.

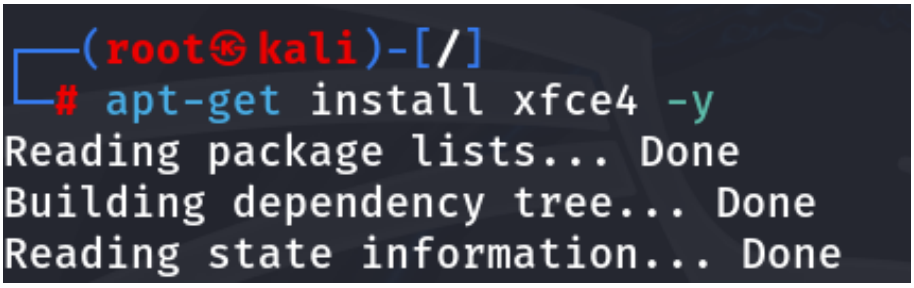
Hands-on Experience – $XDG_CURRENT_DESKTOP

-We can display the current desktop environment by using **echo $XDG_CURRENT_DESKTOP** command.



- In this case, the desktop environment is *GNOME*.

-We can install different desktop environments into our system by using **<package manager name> install <environment_name>** command.



- In this case, name of the package manager is *apt* (advance packaging tool) and installed desktop environment is *XFCE*.

Managing Services

- **Services:** Services are programs or processes that run in our system *at all times*, from booting up until shutting down. Most services are implemented in the form of *daemons*. Daemons are background processes that run *without human intervention* and can start at boot, by applications or manually.
- **init:** It's the backend service that controls when and how services are started.
- **Init Daemon:** It's a configuration file that initiates the processes listed in it. This file prepares our system to run the required software for us. Init Daemon is going to run continuously until system is shut down.

-System initialization begins when the kernel loads and involves loading the operating system components.

- **systemd:** The systemd software is a *modern init method* and is the dominant approach in modern Linux distributions. It offers improvements such as parallelization, control groups (cgroups) for process tracking instead of PIDs etc.
- **systemd unit files:** Unit files are used by systemd utility to manage system resources. These files define how systemd handles various system resources.
- **Systemd Targets:** The systemd utility uses targets to represent specific modes of operation. Targets define different system states. Targets group unit configuration files.

- **systemctl:** This command is used to *control system*. It can *view*, *enable*, *disable*, *start*, *stop* and *restart services*. The systemctl command can also be used to change the system's default target.
- **hostnamectl:** This command is used to *view/change the system's hostname* and *system information*.
- **SysVinit:** SysVinit is an *older init method* with **runlevels** that has been largely *replaced by* ***systemd***. Runlevels control the system's state and determine which daemons should run.
- **telinit:** This command *switches the current runlevel of the system*.
- **runlevel:** This command *prints the previous* and *current runlevels* of the system, each of them separated by a space.
- **/etc/inittab file:** This file stores information about *system initialization* if the system is using *SysVinit*. It defines runlevels and actions to take when changing runlevels.
- **/etc/init.d/ directory:** This directory stores *initialization scripts for services* and controls how they should start.
- **chkconfig:** This command *controls services in each runlevel* and can also enable/disable services during system startup.
- **service:** This command controls *SysVinit services* through SysVinit scripts.

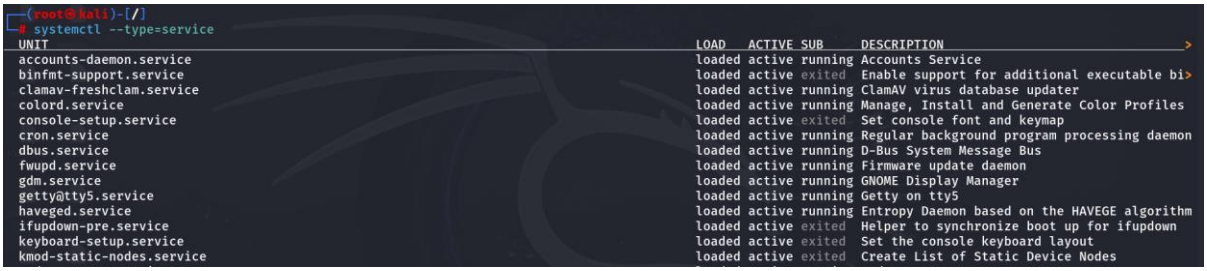
Hands-on Experience – systemctl

-To find out the init method of the system, we can search through the processes. If we come across something that includes '**init'**, it indicates that the system is using **SysVinit**; if it includes **'systemd'**, that would be **systemd**.

```
┌──(root㉿kali)-[/]
└─# ps -e | grep -i init

┌──(root㉿kali)-[/]
└─# ps -e | grep -i systemd
      1 ?        00:01:02 systemd
    748 ?        00:00:03 systemd-logind
   1475 ?        00:00:03 systemd
  23793 ?        00:00:22 systemd-journal
 112984 ?        00:00:01 systemd-timesyn
 169778 ?        00:00:01 systemd-udevd
1474766 ?        00:00:00 systemd-network
```

- In this case, this system is using **systemd** as init method.

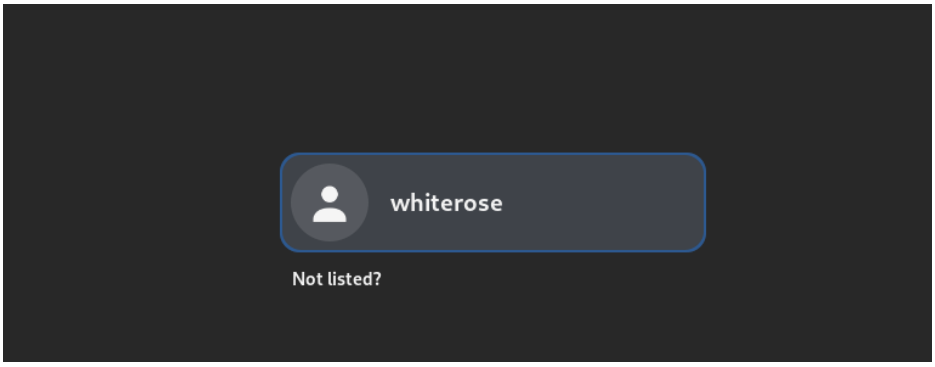
-In order to list all the currently running services on the system, we can use **systemctl --type=service** command.



-We can switch to the multi-user target by using **systemctl isolate multi-user.target** command. Remember that we have to go to the multiuser.target first before getting to the graphical target.

- After executing the command, we need to login on the CLI (command line interface).

-To switch back to GUI (graphical user interface) from command line interface (multiuser.target) which is covered above, we need to imply **systemctl isolate graphical.target** command.



- As we can see, the GUI is back after executing the command.

-To set a service to run as default when the system is getting boot up, we can use **systemctl set-default <service-name>** command.

```
┌──(root㉿kali)-[/home/whiterose]
└─# systemctl set-default graphical.target
Created symlink '/etc/systemd/system/default.target' → '/usr/lib/systemd/system/graphical.target'.
```

- In this case, graphical target (GUI) was set to default.

-We can check the status of a service by using **systemctl status <service-name>** command.

```
┌──(root㉿kali)-[/home/whiterose]
└─# systemctl status tor
○ tor.service - Anonymizing overlay network for TCP (multi-instance-master)
     Loaded: loaded (/usr/lib/systemd/system/tor.service; disabled; preset: disabled)
     Active: inactive (dead)
```

- In this case, tor service (The Onion Router) was inactive.

-We can start a service by using **systemctl start<service-name>** command.

```
┌──(root㉿kali)-[/home/whiterose]
└─# systemctl start tor

┌──(root㉿kali)-[/home/whiterose]
└─# systemctl status tor
● tor.service - Anonymizing overlay network for TCP (multi-instance-master)
     Loaded: loaded (/usr/lib/systemd/system/tor.service; disabled; preset: disabled)
     Active: active (exited) since Tue 2024-09-03 12:43:24 CEST; 3s ago
 Invocation: 3098641028344c8ab076109820c795d2
    Process: 5243 ExecStart=/bin/true (code=exited, status=0/SUCCESS)
   Main PID: 5243 (code=exited, status=0/SUCCESS)

Sep 03 12:43:24 kali systemd[1]: Starting tor.service - Anonymizing overlay network for TCP (multi-instance-master)...
Sep 03 12:43:24 kali systemd[1]: Finished tor.service - Anonymizing overlay network for TCP (multi-instance-master).
```

- In this case, tor service was started.

-We can stop a service by using **systemctl stop <service-name>** command.

```
┌──(root㉿kali)-[/home/whiterose]
└─# systemctl stop tor

┌──(root㉿kali)-[/home/whiterose]
└─# systemctl status tor
○ tor.service - Anonymizing overlay network for TCP (multi-instance-master)
     Loaded: loaded (/usr/lib/systemd/system/tor.service; disabled; preset: disabled)
     Active: inactive (dead)

Sep 03 12:43:24 kali systemd[1]: Starting tor.service - Anonymizing overlay network for TCP (multi-instance-master)...
Sep 03 12:43:24 kali systemd[1]: Finished tor.service - Anonymizing overlay network for TCP (multi-instance-master).
Sep 03 12:44:57 kali systemd[1]: tor.service: Deactivated successfully.
Sep 03 12:44:57 kali systemd[1]: Stopped tor.service - Anonymizing overlay network for TCP (multi-instance-master).
```

- In this case, the tor service was stopped.

Process Issues

-Processes move through a life cycle, from *creation* until *termination*. There are five different process states that a process can be in during this life cycle.

1. **Running State:** The process is currently being executed in user space or kernel space.
2. **Interruptible Sleep State:** The process relinquishes access to the CPU and waits to be reactivated by the scheduler.
3. **Uninterruptible Sleep State:** The process will only wake when the resource it's waiting for is made available to it.
4. **Zombie State:** A process was terminated but not yet released by its parent process so it cannot accept a kill signal.
5. **Stopped State:** The process was stopped by a debugger or through a kill signal.

- **Process ID (PID):** PID is a non-negative integer that is used to *identify a process* and increases for each new process started. Each process has its own PID value. The init daemon always has a PID value of 1 and it is the parent of every other process.
- **pgrep:** This command *displays the PID of processes* that match the pattern. The pgrep command is useful when identifying the PID of processes based on various criteria.
- **pidof:** Another way of finding PID of a process is to use this command. The pidof command allows us to find the PID of a named running program.
- **ps:** This command *displays the process table* which summarizes the current running processes on the system. It has several options. Such as, **a** option lists all user-triggered processes, **-e** option lists all processes, **-l** option lists processes using a long-listing format, **u** option lists processes along with the username and start time, **r** option exclude processes that are not running currently, **x** option includes processes without a terminal.
- **top:** This command *lists all running processes* and provides real-time status. The top command allows us to prioritize, sort and terminate processes in interactive way.

- **htop:** htop is a *newer version of top*. It's also an interactive, system monitor, process viewer, and process manager. It's generally not installed on many Linux distributions.
- **systemd-analyze:** This command *retrieves performance statistics for boot operations*. For process management and troubleshooting, **blame** is the most relevant subcommand.
- **lsof:** This command *prints a list of all files that are currently opened to all active processes*. By using the lsof command, we can identify the offending process for termination.

-Processes are prioritized based on a number from -20 to 19, called a **nice or niceness value**.

- **nice:** This command is used to start a new process with a *specific nice value*. To do this, we need to use **-n** option, which increments the nice value by the provided integer.
- **renice:** This command *alters the scheduling priority of an already running process*. The **-n** option specifies the new nice value for a running process.

-We can run the commands in background, and by using this way, we can execute the commands but not consume our shell. To do this, we can use **fg** (foreground) or **bg** (background) commands.

- **fg:** This command brings a job to the *foreground*. Syntax of this command is **fg %<job ID>**.
- **bg:** This command pushes a job to the *background*. Syntax of this command is **bg %<job ID>**.

- **CTRL + Z:** This shortcut *halts a job temporarily* to allow to use of the **bg** command.
- **&:** This sign starts a command *running in the background* when it's added at the end of a command.
- **jobs:** This command *lists out all jobs* either in foreground or background.
- **CTRL + C:** This shortcut *force quits a running command* on the terminal.
- **CTRL + D:** This shortcut *logs out the current user session* onto the terminal.
- **nohup:** This command *prevents a process from ending when the user logs off*.

- **kill:** This command *sends any specified signal to one or more processes*.
- **pkill:** This command *sends any specified signal to processes based on a matching pattern*.
- **killall:** This command *sends any specified signal to all processes matching the name specified*.

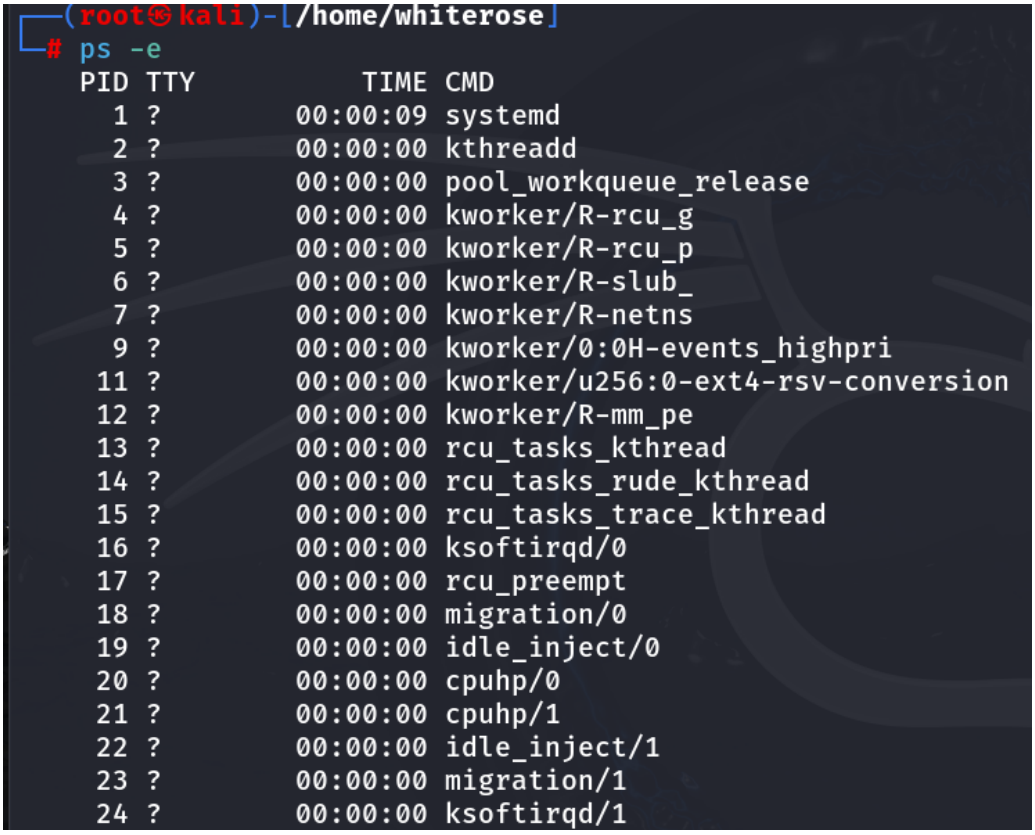
-As a user, we can only kill our own processes. However, as root, we can kill any process we'd like.

Hands-on Experience – ps, pgrep, &, kill, lsof, systemd-analyze

-We can display the processes runing on the shell environment by using **ps** command.

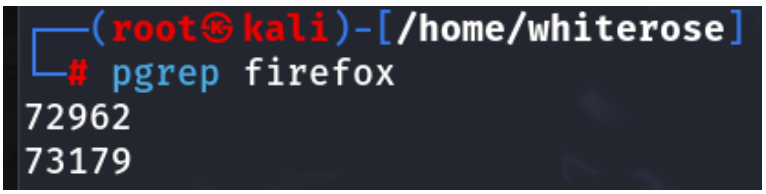
- In this case, *Z Shell*, *sudo*, *su* and *ps* commands are running on this shell by user.

-To see each process on the system, **ps** command with **-e** parameter is used.



- As mentioned above, the *systemd* has PID value as 1 at the top of the output.

-If we want to learn the PID of a specific process, we can use **pgrep** command.

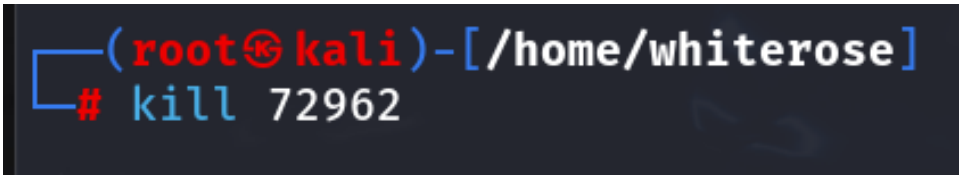


- In this case, PID of *Firefox* is displayed on the screen.

-If we want to run a command in the background, we can use **Ampersand** (**&**) sign at the end of the command.

- In this case, Firefox has been initialized in the background.

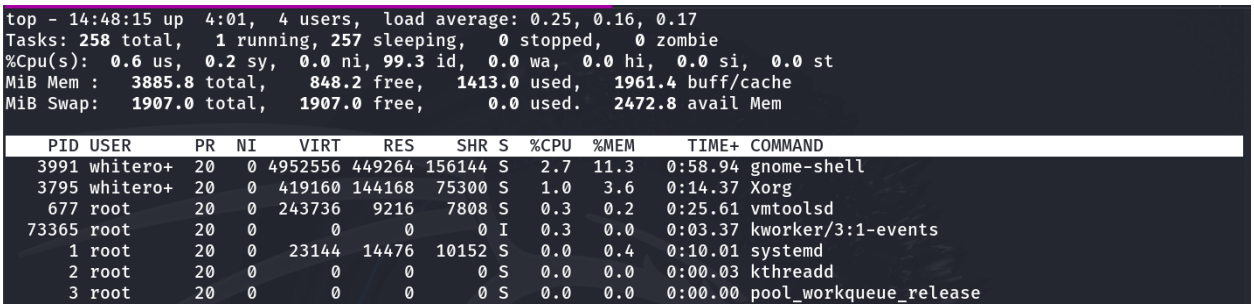
-We can terminate a process by using **kill** command.



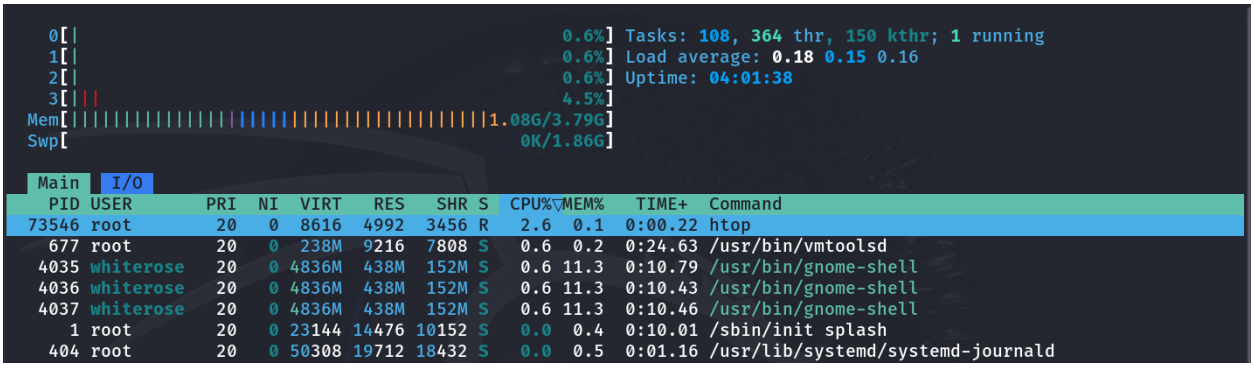
- In this case, the provided PID was Firefox's PID, and it's successfully terminated.

-We can display processes and their memory consumptions on interactive interface by using **top** and **htop** commands.

1- top command



2 - htop command



- The *htop* utility shows the running processes in more human-readable and modern format.

-We can display the files that are currently being ran, and the processes that are running them by using **lsof** command.

```
┌──(root㉿kali)-[/home/whiterose]
└─# lsof
lsof: WARNING: can't stat() fuse.gvfsd-fuse file system /run/user/1000/gvfs
      Output information may be incomplete.
lsof: WARNING: can't stat() fuse.portal file system /run/user/1000/doc
      Output information may be incomplete.
COMMAND     PID  TID TASKCMD              USER   FD      TYPE             DEVICE  SIZE/OFF       NODE NAME
systemd       1                          root  cwd       DIR              8,19      4096          2 /
systemd       1                          root  rtd       DIR              8,19      4096          2 /
systemd       1                          root  txt       REG              8,19    104832    1885177 /usr/lib/systemd/systemd
systemd       1                          root  mem       REG              8,19    407568    1845327 /usr/lib/x86_64-linux-gnu/libbpf.so.1.4.5
systemd       1                          root  mem       REG              8,19    813048    1831702 /usr/lib/x86_64-linux-gnu/libzstd.so.1.5.6
systemd       1                          root  mem       REG              8,19   5706872    1831722 /usr/lib/x86_64-linux-gnu/libcrypto.so.3
systemd       1                          root  mem       REG              8,19    113248    1832776 /usr/lib/x86_64-linux-gnu/libelf-0.191.so
systemd       1                          root  mem       REG              8,19    194552    1836163 /usr/lib/x86_64-linux-gnu/liblzma.so.5.6.2
systemd       1                          root  mem       REG              8,19    112632    1837543 /usr/lib/x86_64-linux-gnu/libkmod.so.2.4.2
systemd       1                          root  mem       REG              8,19    121280    1831714 /usr/lib/x86_64-linux-gnu/libz.so.1.3.1
systemd       1                          root  mem       REG              8,19    633480    1831793 /usr/lib/x86_64-linux-gnu/libpcre2-8.so.0.11.2
systemd       1                          root  mem       REG              8,19    206776    1832709 /usr/lib/x86_64-linux-gnu/libcrypt.so.1.1.0
```

-We can analyze the boot order by using **systemd-analyze** command. Additionally, we can see how long it took for each service to start by adding **blame** subcommand.

1- systemd-analyze

```
┌──(root㉿kali)-[/home/whiterose]
└─# systemd-analyze
Startup finished in 5.457s (kernel) + 35.061s (userspace) = 40.519s
graphical.target reached after 9min 10.451s in userspace.
```

2 - systemd-analyze blame

```
┌──(root㉿kali)-[/home/whiterose]
└─# systemd-analyze blame
1min 36.734s locate.service
     43.581s chkrootkit.service
     24.946s dpkg-db-backup.service
     22.741s plymouth-quit-wait.service
     18.693s man-db.service
      6.133s exim4-base.service
      4.411s logrotate.service
      2.666s thin.service
      2.442s apt-daily-upgrade.service
      2.177s polkit.service
      2.160s NetworkManager.service
      2.027s systemd-journal-flush.service
      1.997s Additional-Tools.mount
      1.913s dev-sdb3.device
      1.680s dbus.service
```

-We can display the priority of processes in the system by using **ps** command with **xl** options.

```
F   UID     PID    PPID PRI  NI    VSZ   RSS WCHAN  STAT TTY        TIME COMMAND
4     0       1       0  20   0  23144 14476 do_epo Ss   ?          0:10 /sbin/init splash
1     0       2       0  20   0      0     0 kthrea S    ?          0:00 [kthreadd]
1     0       3       2  20   0      0     0 kthrea S    ?          0:00 [pool_workqueue_release]
1     0       4       2   0 -20      0     0 rescue I<   ?          0:00 [kworker/R-rcu_g]
1     0       5       2   0 -20      0     0 rescue I<   ?          0:00 [kworker/R-rcu_p]
1     0       6       2   0 -20      0     0 rescue I<   ?          0:00 [kworker/R-slub_]
1     0       7       2   0 -20      0     0 rescue I<   ?          0:00 [kworker/R-netns]
1     0       9       2   0 -20      0     0 worker I<   ?          0:00 [kworker/0:0H-events_highpri]
1     0      11       2  20   0      0     0 worker I    ?          0:00 [kworker/u256:0-ext4-rsv-conversion]
1     0      12       2   0 -20      0     0 rescue I<   ?          0:00 [kworker/R-mm_pe]
1     0      13       2  20   0      0     0 rcu_ta I    ?          0:00 [rcu_tasks_kthread]
1     0      14       2  20   0      0     0 rcu_ta I    ?          0:00 [rcu_tasks_rude_kthread]
```

- On the left-hand side of the output, we can see niceness values. By default, nice values equal zero. When prioritizing tasks keep in mind that the lower the priority the lower the number.

CPU and Memory Issues

-Common CPU issues are *CPU Underperformance, Overloaded CPU, Non-functional Cores*.

- **/proc/cpuinfo file:** This file generally contains *information about the CPU*. Inside of the file, there are some useful fields like *processor*, *vendor_id*, *model name*, *cpu MHz*, *cache size*, *cpu cores*, *flags* etc. Based on the information in /proc/cpuinfo file, the CPU characteristics and performance related issues can be identified.
- **uptime:** This command *displays the time from when a system started running*. The **load average** field in the output of this command is the most relevant in CPU troubleshooting.
- **sar:** This command *displays system usage reports based on data collected from system activity*.
- **lscpu:** This command *displays information about the CPU architecture*.

-In addition to CPU issues, we can have issues with memory. Common memory issues are followed by: *Not enough total memory for all processes or new processes, Processes unable to access memory, Processes accessing too much memory, System cannot access files from cache/buffer*.

- **/proc/meminfo file:** This file contains information about *system memory usage*. It includes useful fields like *MemTotal*, *MemFree*, *Cached*, *SwapTotal*, *SwapFree*, *Dirty*, *Writeback* etc.
  - ***Buffers:*** This field in the /proc/meminfo file indicates memory that has been assigned to a specific block device, like an HDD. It can also be used to cache file system metadata.
  - ***Cached:*** This field in the /proc/meminfo file is similar to Buffers field but instead of storing file metadata. It is going to store the actual contents of the file.
- **free:** This command *parses /proc/meminfo file for memory usage statistics*. Output of this command includes following fields: *total*, *used*, *free*, *shared*, *buffered*, *cached* and *available memory*.

-Memory can also be cached, and it provides to store data temporarily. So, the data can be accessed more *quickly*.

- **lsmem:** This command *lists the ranges of available memory* with their online status.
- **vmstat:** This command *displays statistics about virtual memory*, as well as *process*, *CPU*, and *I/O statistics*. It's recommended to add delay value in order to get more accurate report when running the vmstat command.
- **Out-of-Memory (OOM) Killer:** It's a Linux kernel's feature to *determine processes to kill when the system is extremely low on memory*. The OOM Killer will continue to kill processes until enough free memory is made. Rather than killing processes randomly, the OOM Killer leverages an algorithm that assigns each process on OOM score.

-Configuration of swap space can alleviate memory-related issues. Swap Space is an essential thing to handle memory shortages. Swap files are *dynamic files*. Swap space can be categorized as three types.

1. ***Device Swap Space:*** Used to run large applications.
2. ***File System Swap Space:*** An emergency resource when the available swap space runs out.
3. ***Pseudo Swap Space:*** Enables large applications to run on computers with limited RAM.

- **Swap Partition:** Swap Partition is an *area of virtual memory on a storage device that complements the physical RAM in the computer*.
- **mkswap:** This command is used to *create swap space* on a storage partition.
- **swapon:** This command is used to *activate a swap partition* on a specified device.
- **swapoff:** This command is used to *deactivate the swap partition* on a specified device.

Hands-on Experience – uptime, free, vmstat

-We can display CPU information by reading **/proc/cpuinfo** file.

```
┌──(root㉿kali)-[/home/whiterose]
└─# cat /proc/cpuinfo
processor       : 0
vendor_id       : AuthenticAMD
cpu family      : 25
model           : 80
model name      : AMD Ryzen 5 5600H with Radeon Graphics
stepping        : 0
microcode       : 0xffffffff
cpu MHz         : 3293.728
```

-We can display the uptime of CPU, average amount of processes that are being active or waiting to be used by using **uptime** command.

```
┌──(root㉿kali)-[/home/whiterose]
└─# uptime
 12:20:15 up  1:19,  4 users,  load average: 0.14, 0.14, 0.14
```

-We can display memory information by reading the **/proc/meminfo** file.

```
┌──(root㉿kali)-[/home/whiterose]
└─# cat /proc/meminfo
MemTotal:        3979052 kB
MemFree:         1890188 kB
MemAvailable:    2588176 kB
Buffers:          265744 kB
Cached:           579344 kB
SwapCached:            0 kB
Active:          1288636 kB
Inactive:         376008 kB
Active(anon):     844044 kB
Inactive(anon):        0 kB
Active(file):     444592 kB
Inactive(file):   376008 kB
Unevictable:          16 kB
Mlocked:              16 kB
SwapTotal:       1952764 kB
SwapFree:        1952764 kB
```

-We can display total/free memory and some other information by using **free** command with **-h** option for human-readable view.

```
┌──(root㉿kali)-[/home/whiterose]
└─# free -h
               total        used        free      shared  buff/cache   available
Mem:           3.8Gi       1.4Gi       1.8Gi        23Mi       1.0Gi       2.4Gi
Swap:          1.9Gi          0B       1.9Gi
```

-We can display statistics about virtual memory, CPU and I/O by using **vmstat** command. As previously mentioned, it's recommended to add delay value. Because by default, vmstat command only displays the last information from the most recent reboot.

```
┌──(root㉿kali)-[/home/whiterose]
└─# vmstat 5 3
procs -----------memory---------- ---swap-- -----io---- -system-- --------cpu-------
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st gu
 0  0      0 1841640 265912 759840    0    0   159    16  199    0  0  0 99  1  0  0
 0  0      0 1841640 265912 759840    0    0     0     6  284  444  0  0 99  0  0  0
 0  0      0 1841640 265920 759832    0    0     0     2  200  299  0  0 100  0  0  0
```

- In this case, **vmstat 5 3** command returns 3 queries in every 5 seconds.

Devices

Identifying the Types of Linux Devices

- **Device Drivers:** Device drivers act as an *interface between the operating system and hardware devices*.
- **Thin Clients:** Thin Clients are *lightweight devices that are connected to more powerful servers*. Servers handle processing and storing data. Basically, thin client acts as a user interface.
- **Universal Serial Bus (USB):** USB connects various peripherals such as *input devices, external storage devices* to computers. Plug-and-play technology allows devices to configure themselves automatically. Linux registers USB storage devices attached to the system in **/dev/sd<letter><number>** format.
- **Wireless Devices:** Wireless devices *transmit* and *receive signals over the air*. For instance, Wi-Fi is used for wireless LAN connections, Bluetooth is used for personal area networking (PAN), and NFC is used for close-range data sharing.
- **Video and Audio Devices:** Video and audio devices are input/output peripherals that are connected to client systems. Input devices include webcams, cameras and microphones whereas output devices include monitors, speakers, and headphones.
- **Network Adapter:** Network adapters provide *network connectivity* to computer and act as interface.
- **Network Interface Card (NIC):** Network Interface Card (NIC) is a device that provides an *interface* with which hosts exchange data over a network.
- **General-Purpose Input/Output (GPIO):** GPIO pins on a circuit board that has no designated purpose. GPIO is controlled through software.
- **Serial AT Attachment (SATA):** SATA is a *storage bus interface* for connecting storage devices to traditional computers.
- **PCI:** PCI is a parallel interface bus. PCI supports raw data rates up to 133 MBPS (MB per second).

- **Small Computer System Interface (SCSI):** SCSI is a *computer bus interface* for connecting devices to computers. SCSI is most commonly used for storage.
- **Serial Attached SCSI (SAS):** SAS uses a *serial interface for high-speed storage connections*.
- **Host Bus Adapter (HBA):** HBA is a hardware component that connects a host system to a storage device.
- **Peripheral Component Interconnect (PCI):** PCI is used as an expansion bus for attaching peripheral devices.
- **PCI-express (PCIe):** PCIe is a serial interface bus. PCIe supports raw data rates up to 16 GBPS (GB per second). PCIe supports *greater transfer speeds*, is *more reliable* and is *physically smaller than PCI*. PCIe is the dominant expansion bus technology.

Configuring Devices

- **Device Files:** Device files represent information about hardware devices and settings. Devices files are located in various directories.

-**/proc/** directory contains *information that reported by the kernel*. **/proc/devices** contains list of device drivers that the kernel is currently running.

-**/sys/** directory is a virtual file system that focuses on creating a *hierarchical view of device information*. **/sys/devices/** directory includes files that show *details about specific device*.

-**/dev/** directory enables the *system* and *users to access devices*.

-**/etc/** directory contains *configuration files for components that interface with devices*.

- **Hot-Pluggable:** Hot-pluggable devices can be *added* or *removed without booting*. They are detected by the system. On the other hand, cold-pluggable devices are not detected by the system.
- **udev:** udev utility *handles module loading for hot/cold-pluggable devices*.
- **/etc/udev/rules.d/ directory:** This directory is used to *configure rules for udev functions*. Additionally, this directory is used for local administration of udev.

- **/usr/lib/udev/rules.d/ directory:** This directory contains rules generated by the system.
- **udevadm:** This command is used to manage udev. It includes subcommands. Such as, **info** subcommand views the device's vendor ID, product ID, and serial number, **control** subcommand modifies the running state of udev, **trigger** subcommand executes rules for device events, **monitor** subcommand watches for kernel and udev events and **test** subcommand simulates udev events.

-Printers often come with software utilities, but their compatibility with Linux may vary. Major vendors provide drives for Linux. There is an open-source utility for printers called CUPS.

- **Common Unix Printing System (CUPS):** CUPS is a *print management system for Linux*. CUPS also enables a computer to function as a print server.
- **lpr:** This command submits files for printing.

Hands-on Experience – lsblk, udevadm, lpr

-Assume that we have a USB or external HDD connected to the system. In order to recognize it, we can use **lsblk** command.

```
┌──(root㉿kali)-[/home/whiterose]
└─# lsblk
NAME                  MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
sda                     8:0    0    60G  0 disk
├─sda1                  8:1    0   953M  0 part /boot
├─sda2                  8:2    0   1.9G  0 part [SWAP]
├─sda3                  8:3    0  35.4G  0 part /
└─sda4                  8:4    0  21.8G  0 part /home
sdb                     8:16   0    10G  0 disk
├─sdb1                  8:17   0     2G  0 part
│ └─extra-additional 254:0    0     2G  0 lvm  /Additional/PDFs
└─sdb2                  8:18   0     2G  0 part
  ├─extra-additional 254:0    0     2G  0 lvm  /Additional/PDFs
  └─extra-tools      254:1    0     1G  0 lvm  /Additional/Tools
sdc                     8:32   0 931.5G  0 disk
└─sdc1                  8:33   0 931.5G  0 part
sr0                    11:0    1   3.9G  0 rom
```

- In this case, it's named as **sdc**.

-We can get detailed information about a particular device in the system by using **udevadm** command with **info** subcommand.

```
┌──(root㉿kali)-[/home/whiterose]
└─# udevadm info /dev/sdc
P: /devices/pci0000:00/0000:00:11.0/0000:02:03.0/usb1/1-1/1-1:1.0/host3/target3:0:0/3:0:0:0/block/sdc
M: sdc
U: block
T: disk
D: b 8:32
N: sdc
L: 0
S: disk/by-diskseq/7
S: disk/by-id/usb-WD_Elements_25A2_57585731413638314C454830-0:0
S: disk/by-id/wwn-0x50014ee0af3d667c
S: disk/by-path/pci-0000:02:03.0-usbv2-0:1:1.0-scsi-0:0:0:0
S: disk/by-path/pci-0000:02:03.0-usb-0:1:1.0-scsi-0:0:0:0
S: disk/by-id/ata-WDC_WD10JMVW-11AJGS4_WD-WXW1A681LEH0
Q: 7
E: DEVPATH=/devices/pci0000:00/0000:00:11.0/0000:02:03.0/usb1/1-1/1-1:1.0/host3/target3:0:0/3:0:0:0/block/sdc
E: DEVNAME=/dev/sdc
E: DEVTYPE=disk
```

- In this case, detailed information about **/dev/sdc** device has been displayed.

-We can easily convert text files to pdf files by using **lpr** command. But before applying this command, we need to install **printer-driver-cups-pdf** (CUPS) package from our package manager.

```
┌──(root㉿kali)-[/home/whiterose]
└─# apt install printer-driver-cups-pdf -y
The following packages were automatically installed and are no longer required:
  gkbd-capplet      libfsapfs1t64       libgnomekbd8   libre2-10        libx265-199         python3-oauth2client
  libavfilter9      libfsntfs1t64       libjxl0.7      librecode0       libxklavier16       python3-regex
  libbde1t64        libfvde1t64         libplacebo338  libroc0.3        python3-fasteners
  libclamav11t64    libgeos3.12.1t64    libpostproc57  libsigscan1t64   python3-monotonic
  libdisplay-info1  libgnomekbd-common  librav1e0      libsvtav1enc1d1  python3-nltk
```

- In this case, package manager is **apt**. After installation, go to Settings>Printers and set printer (PDF) as default.

```
┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# lpr list.txt
```

- Use the **lpr** command with the file we'd like to convert to PDF. In this case, it's **list.txt** file.

Monitoring Devices

- **lsdev:** This command *gets hardware information from the /proc/ directory* and *displays* them.
- **/proc/interrupts file:** This file *lists each CPU core* and its *associated interrupt requests (IRQ)*.
- **Interrupt Requests (IRQ):** It's a signal sent by a device to the processor.
- **/proc/ioports file:** This file *lists input/output ports* and the mapped hardware devices.
- **/proc/dma file:** This file lists all Industry Standard Architecture (ISA) Direct Memory Access (DMA) channels on the system.
- **ISA DMA:** A hardware controller that supports legacy technology like floppy disks.
- **lsusb:** This command *shows USB device information*. The lsusb command scans the */dev/bus/usb/* directory for showing information. It has several options. Such as, **-v** option for verbose output, **-s** option filters by bus, **-d** filters by vendor.
- **lspci:** This command *displays information about devices connected to the PCI/PCIe busses*. Default output includes slot address, device class, vendor, and device names.
- **lpq:** This command *shows the status of the print queue*. By default, it shows job rank, owner, files and job size. Additionally, it provides us to update the report at specified intervals with **+interval** option.
- **lprm:** This command removes job/jobs from a printer's queue.
- **lsblk:** The lsblk command was mentioned in the *Managing File Systems* section. It can also be used in *monitoring devices*. The lsblk command identifies block storage devices and checks if they are recognized, partitioned or mounted.
- **dmesg:** This command was also mentioned previously in the *Monitoring Kernel Modules* section. It can also be used in *monitoring devices*. The dmesg command prints all messages sent to kernel's message buffer after booting, including device driver messages.

Hands-on Experience – lspci, lsusb, lpq, lprm

-We can display information about devices connected to the PCI/PCIe busses by using **lspci** command.

```
┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# lspci
00:00.0 Host bridge: Intel Corporation 440BX/ZX/DX - 82443BX/ZX/DX Host bridge (rev 01)
00:01.0 PCI bridge: Intel Corporation 440BX/ZX/DX - 82443BX/ZX/DX AGP bridge (rev 01)
00:07.0 ISA bridge: Intel Corporation 82371AB/EB/MB PIIX4 ISA (rev 08)
00:07.1 IDE interface: Intel Corporation 82371AB/EB/MB PIIX4 IDE (rev 01)
00:07.3 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 08)
00:07.7 System peripheral: VMware Virtual Machine Communication Interface (rev 10)
00:0f.0 VGA compatible controller: VMware SVGA II Adapter
00:10.0 SCSI storage controller: Broadcom / LSI 53c1030 PCI-X Fusion-MPT Dual Ultra320 SCSI (rev 01)
00:11.0 PCI bridge: VMware PCI bridge (rev 02)
00:15.0 PCI bridge: VMware PCI Express Root Port (rev 01)
00:15.1 PCI bridge: VMware PCI Express Root Port (rev 01)
00:15.2 PCI bridge: VMware PCI Express Root Port (rev 01)
```

- To see more detailed information, use **-v** option with **lspci** command.

-We can display USB device information by using **lsusb** command.

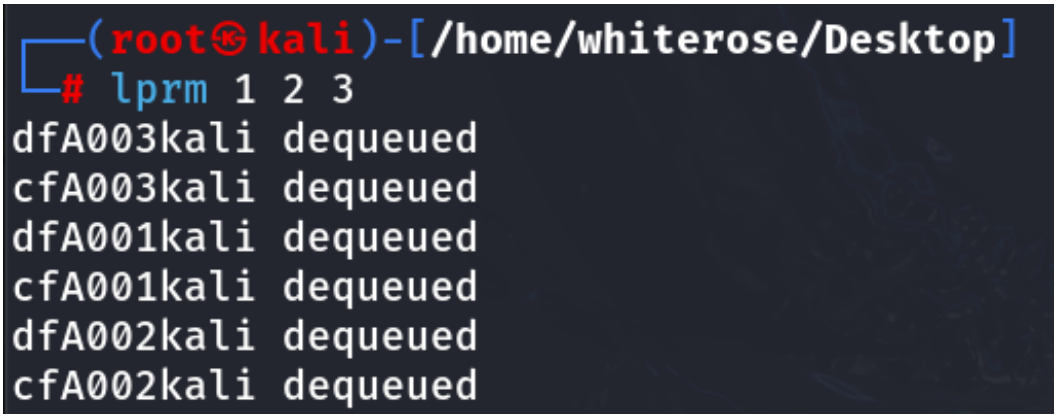
```
┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# lsusb
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 001 Device 002: ID 1058:25a2 Western Digital Technologies, Inc. Elements 25A2
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 002 Device 002: ID 0e0f:0003 VMware, Inc. Virtual Mouse
Bus 002 Device 003: ID 0e0f:0002 VMware, Inc. Virtual USB Hub
Bus 002 Device 004: ID 0e0f:0008 VMware, Inc. Virtual Bluetooth Adapter
```

- To see more detailed information, use **-v** option with **lsusb** command

-We can display printing queue by using **lpq** command.

```
┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# lpq
Warning: no daemon present
Rank    Owner      Job  Files                                 Total Size
1st     root       0    os.txt                                39 bytes
2nd     root       1    os.txt                                39 bytes
3rd     root       2    list.txt                              105 bytes
4th     root       3    list.txt                              105 bytes
```

-We can remove task from the printing queue by using **lprm** command and ID of the printing queue we'd like to remove.



- In this case, printing queues with ID number 1,2, and 3 are dequeued (IDs can be seen with lpq command).

Troubleshooting Hardware Issues

-Issues may affect hardware devices in the system as well. Common hardware issues are followed by: *Keyboard mapping issues*, *Communication port issues*, *Printer issues*, *Memory issues*, *Video issues*, *Storage adapter issues*. Let's explain how to identify and solve each issue listed above.

- **Keyboard Mapping Issues:** Keyboard mapping issues may occur because of *inappropriately configured keyboard layout* or *language*. In such cases, use **localectl status** command to verify and set the appropriate layout. Additionally, remote terminal clients (SSH Clients) like PuTTY enable users to change the effects of keystrokes on the environment.
- **Communication Port Issues:** While facing with Communication port issues, ensure that devices are properly connected in the port, cables aren't loose/damaged, and power is supplied to the system. If there are any, install necessary drivers. Confirm device's compatibility with the bus interface version. For serial devices, configure them to use appropriate serial console (Linux will assign the port at **/dev/ttyS<number>**). If needed, adjust permissions on serial consoles.
- **Printer Issues:** Printer issues may occur depending on physical printer problems such as, paper/ink shortage, paper jams etc. Assuming there is no

problem with printer itself, we may try a few steps. Check printer's manufacturer website. Ensure Linux-compatible drivers are installed. Verify the printer's network connection using **ping** command. Check status of print jobs by using **lpq** command, manage print jobs by using **lprm** command.

- **Memory Issues:** Memory leaks lead to *performance degradation* or *system instability*. To identify and resolve memory leaks, we should follow these steps: Monitor memory usage with **free** command and monitor processes by using **top** command, to identify process related problems and kill them. On the other hand, we might come across *hardware memory* problems. For this type of issues, use **mcelog** command to analyze system logs and look for "*Machine Check Exception*" errors. If the logs contain error-correcting code (ECC) errors, one of the memory modules has failed (indicates a hardware issue). Additionally, we can also perform *stress tests* to identify faulty RAM modules.
- **Video Issues:** Common video issues are consistent *black screen*, *incorrectly display colors* or *multiple monitors not being detected* etc. First ensure the monitor and other devices are properly connected and compatible with the system. If there are no physical problems, check GPU driver (Graphic Card) provided by vendor.
- **Storage Adapter Issues:** Indicators of faulty bus adapters are *poor data transfer speeds*, *less total space available than expected*, *excessive write/read errors*, *inability to read/write*, *device not being detected* etc. The problem might be *HBA* itself or with its connecting interface such as *SCSI* or *SATA*. First ensure that the Host Bus Adapter (HBA) is powered on. Then ensure the device connecting to HBA uses the appropriate interface. For RAID arrays, use **mdadm** command with options to manage RAID arrays.
- **lshw:** This command *lists detected hardware components* on the system and provides device details. It's useful when identifying recognized devices by kernel and reviewing device characteristics. We can specify device classes with **-c** option.
- **dmidecode:** This command *dumps the system's Desktop Management Interface (DMI) table in a human-readable format*. DMI table is an industry standard for tracking information about hardware components. Aware that DMI tables may contain inaccurate information.

- **Automatic Bug Reporting Tool (ABRT):** ABRT reports on problems detected during system runtime, including hardware issues. It collects data like memory dumps and reports to help diagnose problems. ABRT runs as the **abrt** daemon and can be configured using *abrt-cli* or *abrt-gui*.

-While troubleshooting hardware issues, watch these steps:

- Ensure driver support for hardware devices.
- Verify that drivers are installed and loaded.
- Confirm device and software compatibility with Linux.
- Check keyboard layout and language.
- Manage print jobs and queues.
- Monitor memory usage.
- Diagnose and replace faulty RAM.
- Update GPU drivers.
- Check device connections.
- Rescan SCSI buses when needed.
- Use mdadm command for RAID issues.
- Utilize tools like lshw and dmidecode.
- Be cautious of potentially inaccurate DMI data.
- Review crash data reported by ABRT.

Networking

TCP/IP & OSI Model

-The **OSI model** is a *conceptual model* which is *used in communication* over the network. OSI model developed by *ISO* (International Organization for Standardization) in 1984. It stands for *Open System Interconnection*. It consists of *seven layers*; each layer has specific functionality. These layers work collaboratively to *transmit data from one to another*. Let's explain layers one by one. Layers are listed from bottom to top.

1. **Physical Layer (Layer 1):** This layer enables *physical connectivity* (cables, frequencies etc.). Frames are converted into bits and transmitted physically.

2. **Data Link Layer (Layer 2):** This layer enables *physical addressing* (MAC address). Packets are framed and sent to the next device.
3. **Network Layer (Layer 3):** This layer enables *logical addressing* (IP address). Segments are packaged into packets and routed.
4. **Transport Layer (Layer 4):** This layer enables the *reliable transmission of communications using* **TCP** *or* **UDP** based on communication.
5. **Session Layer (Layer 5):** In this layer, *session is established, maintained and torn down* a connection between two devices.
6. **Presentation Layer (Layer 6):** In this layer, *formatting the data* occurs, and *cryptography* is handled (encryption).
7. **Application Layer (Layer 7):** This layer supports *applications and end-users.*

-The **TCP/IP model** is also a *conceptual* and fundamental model for computer networking. It stands for *Transmission Control Protocol/Internet Protocol*. Known as TCP/IP because TCP and IP protocols are two of the foundational protocols in the model. TCP/IP model is used in *modern networks*. TCP/IP model has *four* layers. Let's explain these layers one by one. Layers are listed from bottom to top.

1. **Network Access/Link Layer (Layer 1):** This layer corresponds to *Physical* and *Data Link Layers* in the *OSI model*. This layer is responsible for the *physical transmission* of data over the network (Ethernet, Wi-Fi etc.).
2. **Internet Layer (Layer 2):** This layer corresponds to *Network Layer* in the *OSI model*. This layer is responsible for *routing and addressing packets*. Includes protocols like *IP*, *ARP*, *ICMP*.
3. **Transport Layer (Layer 3):** This layer corresponds to *Transport Layer* in the OSI model. This layer provides end-to-end communication between source and destination. Includes protocols like *TCP* and *UDP*.
4. **Application Layer (Layer 4):** This layer corresponds to *Session*, *Presentation*, and *Application Layers* in the *OSI model*. This layer provides *services to the end-users*. Includes protocols like *HTTP*, *FTP*, *SMTP*.

-In networking, there are terms we may come across besides the OSI model and TCP/IP model. Let's take a look at them.

- **Node:** It refers to devices with an identity on the network. The identity of that node is represented by MAC address, IP address and hostname.
  - **MAC Address:** It's a *physical address* and *unique identifier* which operates at the Data Link Layer (Layer 2) of the OSI model.
  - **IP Address:** It's a *logical* and *unique address* which operates at the Network Layer (Layer 3) of the OSI model.
  - **Hostname:** It's a *human-readable name* of the device. On the internet, domain names meet this need. In our corporate network, hostname meet this need (database server, backup server etc.).

-Apart from terms mentioned earlier, there are network devices and components. These devices need to be properly configured in order to be able to perform their functions on the network. Let's see what these devices/components are.

- **Switch:** A network device which operates in *Data Link Layer* (Layer 2) of the OSI model. A Switch connects devices within a network (Local Area Network) and forward data packets from one to another.
- **Router:** A network device which operates in *Network Layer* (Layer 3) of the OSI model and works with IP addresses. A router connects two or more networks (switches) to each other. So, it allows us to forward and receive data packets from one network (LAN) to another.
- **Media:** Actual path of an electrical signal travelling from one component to another. Typically, we use network cables (twisted pair, coaxial, fiber optic) for this purpose.

-When talking about data, it can evoke different things depending on the layer of TCP/IP or OSI model. If we're operating at the *Network Layer* (Layer 3) of the OSI model, *data* is referred to as **Packet**. When we're operating at the *Data Link* Layer (Layer 2) of the OSI model, *data* is referred to as **Frame**. When we're operating at the *Physical Layer* (Layer 1) of the OSI model, *data* is referred to as **Bit**.

-When dealing with network services, there are two crucial network services called **DNS** and **DHCP**. It's important to understand the logic of these services in order to properly configure network in Linux.

- **Domain Name System (DNS):** DNS is a protocol that provides name resolution, mapping hostnames to IP addresses. DNS is implemented as a database hosted on one or more servers. DNS acts as a phone book. For instance, assume that we'd like to go to '*youtube.com*' but computer will not understand this type of address since it's only dealing with binary, so DNS server converts it to IP address. DNS server stores different types of resource records that are used to resolve names. These records contain name, address and type of record.
    1. ***A:*** An end device IPv4 address.
    2. ***NS:*** An authoritative name server.
    3. ***AAAA:*** An end device IPv6 address.
    4. ***MX:*** A mail exchange record.

-Devices need to be configured to make a connection on the network. It can be done by using two ways: *Static Configuration*, *Dynamic Configuration*. In static configuration, we have to specify network information of a device *manually*. On the other hand, dynamic configuration uses *DHCP protocol*.

- **Dynamic Host Configuration Protocol (DHCP):** DHCP is responsible for the assignment of *IP addresses*, *subnet masks*, *and gateways*. When a host is connected to a network, the DHCP server is contacted, and an address is requested. DHCP server chooses an available address from a configured range of addresses called a pool and assigns (leases) it to the host.

-We've mentioned some crucial services/protocols for communication over the network. But we didn't get into details about IP addresses and other networking terms. Let's delve into it.

- **IPv4 Address:** IPv4 address is a *32-bit* (*4 bytes*) address that is made up of a *Network ID* and *Host ID*. The Network ID must be the same for all devices in the same network. The Host ID is unique to identify a specific host within a network. On the other hand, subnet mask is used to identify the network/host portion of the IPv4 address. IPv4 classes (Class A,B,C,D,E)

determine the number of networks and hosts for each class. **Class A** is between *0.0.0.0* and *127.255.255.255*. **Class B** is between *128.0.0.0* and *191.255.255.255*. **Class C** is between *192.0.0.0* and *223.255.255.255*. **Class D** is between *224.0.0.0* and *239.255.255.255*. **Class E** is between *240.0.0.0* and *255.255.255.255*. Additionally, due to the depletion of IPv4 addresses, there are some reserved IP addresses for internal usage. Also, there are other special IPv4 addresses for specific purposes.

- ***Reserved Ranges:*** In *Class A*, from *10.0.0.0* to *10.255.255.255* range is reserved for private IP addresses (internal networks). In *Class B*, from *172.16.0.0* to *172.31.255.255* range is reserved for private IP addresses. In *Class C*, from *192.168.0.0* to *192.168.255.255* range is reserved for private IP addresses.
- ***Loopback Address:*** This address is used for diagnostics and allows the system to network with itself. *Loopback address* is known as **127.0.0.1** by default.
- ***Link-Local Range:*** This is used for zero-configuration LANs or when the DHCP lease generation process fails (APIPA). Link Local addresses ranged from 169.254.0.0 to 169.254.255.255 for automatic IP configuration.

- **IPv6 Address:** IPv6 address is *128 bits* (*16 bytes*) address. IPv6 provides 2^{128} usable IPv6 addresses. IPv6 is a newer version, fixed weaknesses of IPv4, and has larger address space. It also provides more efficient routing mechanism and built-in encryption.
- **Network Port Numbers:** Port numbers are *numeric values assigned to protocols*. Purpose of port numbers can be explained as ‘humans work with FTP, SSH etc. , while computers need to work by corresponding port number such as 21, 22 etc.’. There are some commonly used port numbers; such as HTTP is 80, HTTPS is 443, FTP is 21, SSH is 22, SMTP is 25, Telnet is 23 etc.

Identifying Linux Server Rules

-Linux servers can be configured in variety of ways to accomplish different services. Let's see the commonly used services that we can configure Linux server.

- **Network Time Protocol (NTP) Service:** NTP is used to *synchronize time settings*. Linux systems can act as NTP servers/clients, ensuring accurate timekeeping for network devices. In addition to NTP, we need software to utilize NTP. One of the commonly used software is *Crony*. Crony is designed to utilize NTP and perform in a large range of conditions.
- **Secure Shell (SSH) Service:** SSH provides an *authenticated* and *encrypted method* of connecting to a remote or local system over the network. It is commonly used for remote administration, but it can also be used to tunnel other different types of traffic through it in a secure way. Linux systems can function as both SSH server and client.
- **Web Servers:** Web servers host the files on websites. It can be either unsecure or secure. HTTP is an unsecure protocol for web servers. It operates through TCP over port 80. On the other hand, HTTPS is a secure and encrypted protocol for web servers. It operates through TCP over port 443. Apart from protocols, web services on Linux are hosted through Apache or Nginx.
- **Certificate Service:** Certificate service provides a way of *guaranteeing identity* whether a user or a server itself. These certificates are known as *digital certificates*, and they are part of the PKI (Public Key Infrastructure). To do this, we have a central server to manage all of these certificates, known as CA (Certificate Authority). Certificate Authority manages the enrollment, approval, expiration, and revocation of certificates. Linux servers can be configured as Certificate Authority Server, enabling secure communication and website authentication.
- **Name Resolution Server/ DNS Service:** DNS service performs resolving hostnames to IP addresses. Also, DNS server may contain records for a company's internal network. Linux systems can function as both DNS servers and clients.

- **DHCP Server:** DHCP server assigns IP addresses, provides subnet masks and default gateways to clients *dynamically*. Linux can serve as a DHCP server/client in order to simplify IP configuration management.
- **Simple Network Management Protocol (SNMP) Service:** SNMP service enables *monitoring* and *management of network devices* (routers, switches etc.). Linux servers can act as SNMP managers/agents, can collect and transmit device performance data.
- **Authentication Service:** Authentication service is used to *verify credentials* and *authentication* of a person when person verifies their identity to an application. Authentication server relies on protocols such as *LDAP* (Lightweight Directory Access Protocol), *Kerberos*, *OAuth* etc. Linux servers can operate as an authentication server.
- **Proxy Server:** Proxy server has both *direct access to the Internet* and *internal network connection*. So, it acts as a person in the middle of the connection. Proxy server is useful when connecting to untrusted resource. Linux can function as a proxy server.
- **Logging Server:** Centralized logging server *collects* and *stores log data* for security monitoring, troubleshooting and analysis purposes. One of the key ways of creating a logging server is to use *syslog* protocol. Linux can be used as centralized log server.
- **Monitoring Service:** Linux offers various monitoring tools to *track system* and *application performance*. Such as, *ApacheTop* provides log file analysis for Apache and connection response times etc.
- **Load Balancing Service:** Load balancing service is used to *distribute inbound connection requests across multiple servers*. Load balancing service provides best performance and prevents crashes. Load balancing service is commonly used in web services.

-Besides these services, also another services available in *internal network*. Such as, *database servers*, *storage area networks* (SAN), *virtualization services*, *VPN services* etc. Additionally, most of these services run on Linux. Each server inside of this cluster is called a *node*. **Node** is a server inside a cluster and can accept client connections.

- **File/Print Service:** File server *stores* and *centralizes user data*; print server *manages printers* connected to network. There are two main types of file

servers: **Samba** and **NFS**. *Samba (Server Message Block)* is a Windows-compatible file sharing system that runs on SMB (Server Message Block). *NFS (Network File Sharing)* is a native Unix/Linux protocol that is used to provide workstations to access files stored on Linux/Unix servers.

- **Database Service:** Database service is used to *store large quantities of data* and provide to make easy queries. There are two basic types of databases inside Linux: **SQL** and **NoSQL**. Such as *PostgreSQL*, *MySQL* etc. SQL database structured databases which use relational tables. In *NoSQL databases*, there is no centralized, organized way and it does not use relational tables. NoSQL database is used in high-capacity systems and fields like Machine Learning, AI etc. *MongoDB* is a NoSQL database example.
- **Virtual Private Network (VPN) Service:** VPN server enables *remote employees to connect to the internal company network* and access internal resources. VPN service allows to have *secure tunnel* when connecting to company's network.
- **Email Service:** Email service is responsible for the distribution of electronic mail (Email). There are two types of email servers: *Sendmail* and *Postfix*. To have sending server, Sendmail is used. To be able to receive email, Postfix is used. When sending email, *Simple Mail Transfer Protocol* (SMTP) is used. When receiving email, *Post Office Protocol* (POP3) is used.

Connecting to a Network

-Computers are connected to a network to be able to communicate with each other. Two or more computers that are connected through network media are called a **computer network**.

-In Linux, we use a device or system hostname to easily recognize the machine within a network.

- **hostnamectl:** This command *allows us to set a hostname* for our machine. For setting hostname, **set-hostname** subcommand is used.

-For a computer to be able to participate in a network, it has to have a valid identity. This covers an *IP address*, a *MAC address* and a *hostname*.

- **IP Configuration:** IP addresses can be configured using either dynamically (DHCP) or statically. In dynamic configuration, IP address, subnet mask, default gateway, DNS server information will be configured *from DHCP server automatically*. But in static configuration, we have to *manually assign* them all. Misconfigurations in these values will result in non-participation in the network.
- **NetworkManager:** This utility allows us to set up the proper configuration of the IP information. Many Linux distributions have NetworkManager utility. NetworkManager includes three different interfaces.
    1. **nmcli:** This is a *command line interface tool for NetworkManager*. It contains lots of subcommands to view and configure network information. Such as, **general status** subcommand shows summary of network connectivity data, **connection show** subcommand views identification information for each NIC, **con up** subcommand enables specified NIC, **con down** subcommand disables specified NIC, and **device status** subcommand displays current status of each NIC.
    2. **nmtui:** This is a *text-based user interface (TUI) for NetworkManager*. We can navigate through the interface using the Tab key, Spacebar, Enter key and arrow keys.
    3. **nmgui:** This is a *graphical user interface (GUI) for NetworkManager*. We can configure IPv4, IPv6 and various network settings.

-When troubleshooting a network connection, very first thing to do is to *view IP address* by using **ifconfig** or **ip** command.

- **ifconfig:** This command *shows the IP address, subnet mask, broadcast ID, MAC address, basic performance information,* and *NIC name*.
- **ip:** This command has largely *replaced the ifconfig command* and *functions as the same*. The ip command is much *more powerful* than ifconfig command. It has several subcommands. Such as, **addr show** subcommand shows the IP address information, **link** subcommand shows the status of each interface, **link set <interface> up** subcommand enables the status of the interface up, **link set <interface> down** subcommand enables the status of the interface down etc.

- **iwconfig:** This command *provides wireless NIC configurations* such as wireless network interfaces, including SSID, frequency, and channel settings.

Hands-on Experience – nmcli, hostnamectl, ifconfig, ip, ethtool

-We can display information related to network and our device by using **nmcli** command. It's a command line interface for NetworkManager utility.

```
┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# nmcli general hostname
kali

┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# nmcli general status
STATE      CONNECTIVITY  WIFI-HW  WIFI     WWAN-HW  WWAN     METERED
connected  full          missing  enabled  missing  enabled  no (guessed)
```

- In this case, first command, which is **nmcli general hostname**, displayed the hostname. Second command, which is **nmcli general status**, displayed the network information. There are bunch of other subcommands available.

-We can display specific device type of information by using **nmcli** command with **connection** subcommand.

```
┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# nmcli connection
NAME                UUID                                  TYPE      DEVICE
Wired connection 1  d94cd00b-3a9d-4d4b-8054-430b1223a75d  ethernet  eth0
lo                  a1699f79-a013-437c-b01f-eb0844003f92  loopback  lo
```

-We can change the hostname by using **hostnamectl** command with **set-hostname** subcommand.

```
┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# hostnamectl set-hostname emre

┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# hostname
emre
```

- In this case, hostname has changed to as **emre**. However, this change is not permanent. Let's find out how to change it permanently.

-To provide the new hostname configured with hostnamectl command persistent, we need to restart the systemd. We can do it by using **systemctl restart systemd-hostnamed.service** command.

```
┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# systemctl restart systemd-hostnamed.service
```

-We can display IP address and other related information by using **ifconfig**. However, keep in mind that the ifconfig command is largely replaced by **ip** command. We'll cover it as well.

```
┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.50.131  netmask 255.255.255.0  broadcast 192.168.50.255
        inet6 fe80::20c:29ff:fe57:b7de  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:57:b7:de  txqueuelen 1000  (Ethernet)
        RX packets 536535  bytes 506434706 (482.9 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 277399  bytes 75899141 (72.3 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 32088  bytes 34061716 (32.4 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 32088  bytes 34061716 (32.4 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:57:b7:de brd ff:ff:ff:ff:ff:ff
    inet 192.168.50.131/24 brd 192.168.50.255 scope global dynamic noprefixroute eth0
       valid_lft 1241sec preferred_lft 1241sec
    inet6 fe80::20c:29ff:fe57:b7de/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

- In this case, both **ifconfig** and **ip addr** commands display same information. But **ip** command has more options.

-We can display specific information about a network interface by using **ip addr show <interface>** command.

```
┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# ip addr show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:57:b7:de brd ff:ff:ff:ff:ff:ff
    inet 192.168.50.131/24 brd 192.168.50.255 scope global dynamic noprefixroute eth0
       valid_lft 1011sec preferred_lft 1011sec
    inet6 fe80::20c:29ff:fe57:b7de/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

- In this case, **ip addr show eth0** command displayed information about eth0 interface.

-We can disable a network interface in two ways: **ifconfig <interface> down** and **nmcli con down <interface>**. First way is an old method (since ifconfig is deprecated) whereas second way is more important to remember for the exam.

-To enable the network interface back, here are the commands: **ifconfig <interface> up** and **nmcli con up <interface>**.

-We can manually assign IP address by using a menu in NetworkManager and to go to that menu, we can use **nmcli con edit <interface>** command. After going into the menu, we are going to use **set ipv4.addresses <IP_address>/<prefix>** and **save** command to save the changes.

```
┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# nmcli con edit Wired\ connection\ 1

===| nmcli interactive connection editor |===

Editing existing '802-3-ethernet' connection: 'Wired connection 1'

Type 'help' or '?' for available commands.
Type 'print' to show all the connection properties.
Type 'describe [<setting>.<prop>]' for detailed property description.

You may edit the following settings: connection, 802-3-ethernet (ethernet), 802-1x, dcb, sriov, ethtool, match, ipv4
, ipv6, hostname, link, tc, proxy
nmcli> set ipv4.addresses 192.168.50.102/24
Do you also want to set 'ipv4.method' to 'manual'? [yes]: y
nmcli> save
Connection 'Wired connection 1' (d94cd00b-3a9d-4d4b-8054-430b1223a75d) successfully updated.
```

- In this case, IP address has changed to **192.168.50.102**. Additionally, we need to **disable** and **enable** the *interface* or **restart** the *NetworkManager* by using **systemctl** command to update IP information.

-To configure DHCP on a network interface, we can use **nmcli con edit <interface>** again to access the menu, then **set ipv4.method auto** and save command to save the changes.

```
┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# nmcli con edit Wired\ connection\ 1

===| nmcli interactive connection editor |===

Editing existing '802-3-ethernet' connection: 'Wired connection 1'

Type 'help' or '?' for available commands.
Type 'print' to show all the connection properties.
Type 'describe [<setting>.<prop>]' for detailed property description.

You may edit the following settings: connection, 802-3-ethernet (ethernet), 802-1x, dcb, sriov, ethtool, match, ipv4
, ipv6, hostname, link, tc, proxy
nmcli> set ipv4.method auto
Do you also want to clear 'ipv4.addresses'? [yes]: y
nmcli> save
Connection 'Wired connection 1' (d94cd00b-3a9d-4d4b-8054-430b1223a75d) successfully updated.
```

- Do not forget to **disable** and **enable** the *interface* or **restart** the *NetworkManager* by using **systemctl** command to update IP information.

-Another tool to get information about NIC is the **ethtool** command.

```
┌──(root㉿kali)-[/home/whiterose/Desktop]
└─# ethtool eth0
Settings for eth0:
        Supported ports: [ TP ]
        Supported link modes:   10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
                                1000baseT/Full
        Supported pause frame use: No
        Supports auto-negotiation: Yes
        Supported FEC modes: Not reported
        Advertised link modes:  10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
                                1000baseT/Full
        Advertised pause frame use: No
        Advertised auto-negotiation: Yes
        Advertised FEC modes: Not reported
        Speed: 1000Mb/s
        Duplex: Full
        Auto-negotiation: on
        Port: Twisted Pair
        PHYAD: 0
        Transceiver: internal
        MDI-X: Unknown
        Supports Wake-on: d
        Wake-on: d
        Current message level: 0x00000007 (7)
                               drv probe link
        Link detected: yes
```

Configuring DHCP and DNS Client Services

-As mentioned before, there are two ways to assign IP address: *statically* and *dynamically*. Static IP configuration involves manually setting IP address, subnet mask, default gateway and DNS address; and it's typically used for corporate networks (network devices, servers, end-users, printers etc.). Dynamic IP configuration relies on DHCP servers to provide IP address and other related information dynamically to client machines. Assuming that we've decided to assign IP addresses dynamically, let's learn how to configure. In this type of scenarios, properly configuring DHCP service is crucial. Additionally, properly configuring DNS service is also crucial under all circumstances.

- **DHCP Configuration:** DHCP servers contain pools of IP addresses and related configuration options for client machines once they connect to the network. When DHCP assigns an IP address to a client, it's called **lease**. DHCP servers configured with a given range of available IP addresses along with additional options like the DNS server's location and default gateway. The DHCP service must be installed on the server and allow client machines to **lease configurations**. Leasing an IP address from DHCP server involves four steps:
    1. ***Discover:*** When client first connects to the network, it will send out **DHCPDISCOVER** message, in order to identify any available DHCP servers on the network.
    2. ***Offer:*** DHCP server replies to DHCPDISCOVER message with **DHCPOFFER** message. This message contains the *IP address* and *subnet mask* to assign, *IP address of DNS server*, *IP address of the gateway* and also the *duration of the lease* (lease time).
    3. ***Request:*** If there are multiple DHCP servers available on the network, the host chooses between them and sends out a **DHCPREQUEST** message.
    4. ***Acknowledge:*** Finally, DHCP server returns **DHCPACK** message to the client.

-In Linux, we can configure DHCP client in **/etc/dhcp/dhclient.conf** file.

- **/etc/dhcp/dhclient.conf file:** This file *enables the configuration of DHCP client settings*, including *timeouts* and *dynamic DNS configurations*. This file is generally edited by using NetworkManager. Whenever we commit changes to the DHCP client configuration inside NetworkManager, it's going to update /etc/dhcp/dhclient.conf file.

-When configuring DNS settings in Linux, there are two ways we can follow: First way is to configure **/etc/hosts** file (includes *static IP addresses*). But this way does not make any sense for browsing online in today's world. Other way is to use *dynamic system of DNS* (**/etc/resolv.conf** file).

- **/etc/hosts file:** This file contains *static IP addresses* and *hostname mappings*. It provides a manual way of resolving names to IP addresses.
- **/etc/resolv.conf:** This file specifies the *IP addresses of DNS servers*. It allows systems to query DNS for name resolution.
- **/etc/nsswitch.conf:** This file defines the *order of name resolution methods*. Basically, it specifies whether to use /etc/hosts file for static resolving or DNS (specified in /etc/resolv.conf) first. By default, /etc/hosts is checked as *first option*, then DNS is checked as *second option*. We can change it inside the /etc/nsswitch.conf file.

-Also remember that *NetworkManager* utility *provides to configure DNS settings* by using its CLI, TUI, and GUI tools.

-In case we'd like to learn the IP address of a website, we can use **dig**, **nslookup**, **ping**, and **host** commands. These commands help us to ensure that the DNS is working properly and also to test name resolution. Additionally, these commands provide some other information about the specified host.

- **whois:** This command *displays information about the owner of a domain name*. Such as, contact information of the owner, mailing address, location etc.

Cloud Technologies

-Many of the cloud servers are built on top of Linux. Cloud technology leverages networking; it provides us with infrastructure, service, platforms, applications etc. It can be done either using network or the Internet. Cloud computing is a relatively new and rapidly growing aspect of the IT industry.

-According to the National Institute of Standards and Technology (NIST), there are five essential characteristics of cloud computing. They are listed as follows:

1. *On-demand self-service*
2. *Broad network access*
3. *Resource pooling*
4. *Rapid elasticity*
5. *Measured service*

-Cloud services indicate flexibility in terms of deployment, scale support, and fault tolerance. There are three primary models of cloud computing with many sub variations of each of them.

- ***SaaS:*** Stands for *Software as a Service*. Provides application to the end-users. Applications are not installed directly, instead delivered over the network.
- ***PaaS:*** Stands for *Platform as a Service*. Includes virtualization of the operating system layer of the environment. Provides services to developers and database administrators.
- ***IaaS*****:** Stands for *Insfrastructe as a Service*. Includes physical devices that are virtualized and owned by a cloud service provider.

-Cloud service can also be categorized based on who owns the cloud environment. This is referred to as public, private, or hybrid cloud infrastructure.

- ***Public Clouds:*** Infrastructure and hardware resources that are *owned by the cloud service providers*, and they are shared among multiple customers. For example, Amazon Web Services (AWS), Microsoft Azure etc. Public cloud services are considered cost efficient. Disadvantage of using public cloud services can be security concerns, since they can be vulnerable to cyber-attacks.

- ***Private Clouds:*** The use of private cloud technologies as an on-premise solution. In a private cloud, *an organization itself owns* and *operates all the servers* and *infrastructure*, and then provides them to the rest of the organization. Using private clouds provides a higher level of security. Since private clouds require dedicated components (hardware, infrastructure), it can be expensive to acquire and maintain.
- ***Hybrid Clouds:*** Hybrid Cloud is a *combination of Public and Private Clouds*. It Enables more effective cost management combined with strict security management. Complexity of using hybrid clouds can be considered as a disadvantage.

Virtualization Technologies

-Basically, cloud technologies rely on virtualization, allowing end-users to run virtual machines on these cloud-based servers.

- **Virtualization:** Virtualization *forms the foundation of cloud computing*. It allows efficient use of hardware, fault tolerance, disaster recovery, and scalability. To run virtual machines on a server, the physical server must have a *Hypervisor* installed.
- **Hypervisor:** Hypervisor is a *virtualization software layer* that provides control between the virtual machines and the physical hardware. There are two types of hypervisors:
  - ***Type 1 Hypervisor:*** Type 1 Hypervisor runs *directly* on the hardware called *bare metal deployment*. It tends to be more efficient because they don't run an entire operating system. Some instances of Type 1 Hypervisors are **VMware ESXi**, **Microsoft Hyper-V**.
  - ***Type 2 Hypervisor:*** Type 2 Hypervisor runs *as a service* on top of the operating system. Type 2 hypervisor requires additional resources to operate since they got operating system installed, then hypervisor, and guest virtual machine with its operating system. Most common Type 2 Hypervisors are **VMware Workstation Pro**, **QEMU**, **Oracle VirtualBox**.

- **Kernel-Based Virtual Machine (KVM):** KVM is an open-source virtualization technology built into Linux. KVM lets us turn Linux into hypervisor that allows a host machine to run multiple, isolated virtual environments called guests or virtual machines. KVM converts Linux into a type-1 (bare-metal) hypervisor. KVM has all of the operating system components (process scheduler, I/O, device drivers etc.) because it's part of the Linux kernel.

-When working with virtual machines, it's crucial to create and use virtual machine templates to make deployments easier and more efficient. Basically, they provide us with automate the deployment of virtual machines. There are different types of template files and formats that we might come across.

- ***Open Virtualization Format (OVF):*** Contains *configuration files*, *packages*, and *settings* for virtual machines and network devices.
- ***JavaScript Object Notation (JSON):*** Used by most programming languages to store information.
- ***YAML Ain't Markup Language (YAML):*** Used to *store configuration information* on the newly deployed virtual machines.
- ***Container Images:*** Used by a specialized type of virtual machine called container.

-Let's define boot process and bootstrapping.

- **Bootstrapping:** Bootstrapping refers to the *initial setup* and *customization of virtual machines* during their first boot. Tools like cloud-init, Anaconda, and Kickstart enable bootstrapping. These tools help automate the installation and configuration of virtual machines.
- **Storage:** Storage in virtualization can be provided as *virtual storage drivers*. These virtual storage drivers are the portions that created across a RAID device or SAN (Storage Area Network). Virtual storage allows more efficient flexibility, fault tolerance, and scalability. Additionally, when deploying a virtual machine, we may be offered the option of configuring thin storage or thick storage. Let's explain what they are.
    - ***Thin Storage:*** It's a virtual storage device file which will grow on demand up to a maximum size. This tends to be more efficient use of drive space but might have slower performance than thick storage.

- ***Thick Storage:*** It reserves the allocated space for use by the virtual device. Assume that we've installed a virtual machine and set up the virtual drive to have one terabyte volume size, it's going to create a one terabyte file immediately, even though we only need ten gigabytes, if we choose to use this *thick provisioning*, during the virtual machine creation. Benefit of *thick provisioning* is the guaranteed space available. Disadvantage is that we instantly used up a lot in disk space, so it's a huge waste.

- **Blocks:** Blocks are small chunks where the data is written in storage device (both physical and virtual).
- **Virtualization Hypervisor:** Virtualization hypervisor can be configured to provide access to networking services in several ways. The virtual machine can be configured with one or more virtual NICs. These NICs can be connected to virtual switches within that hypervisor as well.
  - ***No networking:*** This option simulates a computer that does not have a NIC at all, so there is *no network connectivity* for that virtual machine.
  - ***Internal option:*** This option is where the virtual machine is connected to a virtual switch which permits network communication with other virtual machines connected to same virtual switch. So, this *won't allow* them *to communicate with the host operating system*.
  - ***Private option:*** This option allows the virtual machine to be connected to a virtual switch which permits network communication with other virtual machines and also host operating system. However, there is *no public internet connection* available.
  - ***Public option:*** In this option, virtual machine is connected to virtual switch which permits *network communication with other machines*, *physical NIC* as well as the *public internet*.
- **Network Address Translation (NAT):** NAT provides *virtualized network* functionality in physical networks. It will translate between private IP address ranges and the public internet IP address ranges.

-Virtualized networks may be thought of as *overlay networks*. Additionally, most of the Linux server is running without a GUI to save on memory and processing resources. This is referred to as *Headless Mode*.

- **virsh:** This command *provides interactive shell* that *controls the virtual machines*.
- **libvirt:** This command allows us to build *virtualization solutions*. It's an API (application programming interface) and provides the software building blocks to write virtualization solutions. VMware ESXi, KVM, QEMU hypervisors are built using libvirt.
- **Virtual Machine Manager (VMM):** This utility is used for *managing connectivity to virtual machines in GUI environments*.

Troubleshooting Network Issues

-When encountering a network interface problem, we should follow some steps in order to obtain and troubleshoot the problem itself. Basic step to identify the problem start with checking if the device is powered on or plugged in. Then verify and configure the network interface (it must have appropriate IP, subnet mask, default gateway and name resolution settings). After that, check if network interface is detected by Linux (**ifconfig**, **ip**, **ethtool** etc). If the NIC is not detected by the system, verify the appropriate driver has been installed.

-When encountering a problem in reaching websites, we can follow these steps to obtain the problem. We can ping a destination by hostname and by IP address; if we can ping by hostname and can't ping by IP address, it indicates that system is not properly resolving (DNS issue). We can use **host** or **nslookup** commands to test system's ability to perform DNS. If we come across a high latency issue, we can check latency statistics by using **netstat** command.

-Sometimes, we can have problem with NIC itself. In such cases, data packets can be dropped on the network which causes a loss in data and timeout messages. In this type of problems, replace the NIC or check other point of the connection.

-When trying to troubleshoot an application's performance on Linux, we have to decide which way the communication is going to occur. This can be done using either a local host or Unix sockets. First method known as local host is going to create a full network connection and TCP error-checking to troubleshoot the connection. Second method relies on Unix sockets, and this method is significantly faster than the first method.

-Let's take a look at the utilities for troubleshooting and explain what they do.

- **ping:** This command is used to *test packets between two systems*. It's a *TCP/IP utility*. On the output, receiving **<host>** indicates that connection is successful, **Destination unreachable** indicates that system can't find the path to the destination, and **timeout** indicates that request reached the destination, but no response returned. Keep in mind that the ping command tells us something's wrong, not what is wrong.
- **traceroute:** This command is used to *report the network path between the source and destination*. The process of a packet travelling across from one router to the next is known as **hop**. The traceroute command will give an output for each hop along the path. By using **traceroute** and **tracepath** commands, we can observe routing loops (tracepath command is similar to traceroute).
- **netstat:** This command is used to *gather information about TCP connections on the system*. The netstat command is capable of informing user of existing connections, ports that are set in listening state, NIC information etc.
- **ss:** This command is *newer version of netstat command*. The ss command is an information gathering utility with simpler output and syntax. It stands for *socket state*. On the output, **missing socket** means that the service isn't running, and **closed socket** means that either the client or the server is prematurely terminating that connection.
- **dig:** This command is used for *gathering information* and *testing name resolution*. It's useful when we have DNS issues.
- **nslookup:** This tool is also used for *gathering name resolution information* and *testing name resolution*. This command is available on most Linux distributions as well as Windows and macOS.
- **host:** This command is *capable of gathering information* and *testing name resolution*.
- **route:** This command is used to *view the routing table*. The route command also allows us *to configure routing table* if we need to.
- **nmap:** This tool is a powerful tool used for *network scanning*. It can be used to *identify nodes*, to check for *available services, operating system information, host name, IP address, open ports, MAC address* and much

more on a target host. The nmap command is heavily used by cybersecurity professionals, in order to scan vulnerabilities on a network/system. It stands for *Network Mapper*.

- **Wireshark:** Wireshark is an immensely popular *packet sniffer* and *network analyzer tool*. Basically, it intercepts and displays network traffic. Wireshark is a *GUI-based tool*. This tool is widely used by network admins, cybersecurity professionals and threat actors. Wireshark has the ability to see moving or non-moving packets through NIC.
- **tcpdump:** In addition to Wireshark, there is another tool called tcpdump. It's a *CLI-based tool* which is used for *packet sniffing*. It captures network packets.
- **nc:** This command is used to *test connectivity* and *send data across network connections*. It stands for *netcat*. Netcat can also be used for creating network connections between hosts.
- **iftop:** This command displays *bandwidth usage information* for the system. It helps to identify *bandwidth-hungry applications* and is useful for investigating network slowness.
- **iperf:** This command is used to test the *maximum throughput* of an interface. The iperf command ensures that if the network bandwidth meets expectations.

-Both the iftop and iperf utilities measure throughput, and there is a difference between bandwidth and throughput. **Bandwidth** is *potential* amount of data that we could move through a network connection in a given amount of time. **Throughput** is the amount of data that is *actually* moving through the network.

- **mtr:** This utility is the *combination of ping* and *traceroute* commands. It's used to test the quality of network connections, and it identifies packet loss and network issues. It stands for My Traceroute.
- **arp:** ARP is the Address Resolution Protocol and is used to *resolve IP addresses* to *MAC addresses*. Basically, doing the *conversion from layer two to layer three* (or opposite depending on ARP/Reverse ARP). The arp command itself helps us to discover information about known MAC addresses on a given system. Additionally, the arp command also provides clearing the ARP cache.

Packages and Software in Linux

Package Managers

-Linux makes use of different distributions that have different ways of managing packets. These are known as Package Managers. In Linux environment, package managers are used for software management. Linux distributions rely on two different methods of managing software throughout its lifecycle: Package Managers and Compiling Software.

1. ***Package Managers:*** Package Managers refer to programs that install, update, inventory, and uninstall packaged software.
2. ***Compiling Software:*** It refers to the process of taking source code written in high-level programming language and transforming it into an executable program that can run on the computer. Using compiling software is much more difficult because of the open-source nature of Linux.

-Many Linux applications are modular and depend on *dependencies*. Package managers check dependencies before installing software from the package. On the other hand, if we're manually compiling the software, we need to work through all dependencies on our own. However, compiling the software manually offers customization. In Linux, there are two dominant methods for managing software packages:

1) **RPM:** It stands for *Red Hat Package Manager*. RPM packages have *.rpm* as file extension. Used in distributions like *Red Hat*, *CentOS*, *Fedora*, *Oracle Linux* etc. One of RPM's most useful features is its ability to *inventory software*.

-There is a newer and more advanced package manager than RPM and is also supported by Red Hat derivatives. This is known as YUM (Yellowdog Updater Modified).

- ***YUM (Yellowdog Updater Modified):*** YUM *relies on RPM* and uses *.rpm* package files. It offers a more elegant set of commands and greater flexibility for handling dependencies.

-Additionally, there is also a newer version of YUM. It's known as DNF (Dandified YUM).

- ***DNF (Dandified YUM):*** *Improved version of YUM* and uses fewer resources while still maintaining support for RPM.

-Another package manager we may come across is known as Zypper.

- ***Zypper:*** It's an *openSUSE package manager* that supports *.rpm* packages.

2) **dpkg:** It stands for *Debian Package*. DPKG packages have *.deb* as file extension. Used in *Debian-based* distributions such as Kali Linux.

-On Debian-based distributions, a package manager named *APT* (Advanced Package Tool) is widely used.

- ***APT (Advanced Package Tool):*** More *flexible* and *preferred package manager tool* rather than dpkg in Debian derivatives. It *relies on .deb* packages, much like YUM.

RPM and YUM Packages

-RPM and YUM packages are important because they are used in Red Hat, CentOS and derivatives. Since a lot of Linux servers use Red Hat derivatives, it's crucial to get more knowledge about these packages.

- **rpm:** This command *manages the RPM packages* on Red Hat derivative distributions. It has several options. Such as, **-i** option installs specified package, **-e** option uninstalls specified package, **-v** option enables verbose mode to see more detail, **-h** option prints hash marks to indicate a progress bar, **-V** option verifies the software components of specified package. We can query packages to get information like package version information, dependency information etc. To do that, we need to use several options. Such as, **-qa** option lists all installed software, **-qi** option lists information about specified package, **-qc** option lists the configuration files for specified package. RPM offers two primary ways to update and upgrade: First method is **-U** option and second method is **-F** option. Difference between them is **-F** option freshens installed packages whereas **-U** option will install the package if it doesn't exist.

- **yum:** This command is another way to use in package management. It improves the functionality of rpm, while still using the *.rpm packages* and maintaining an rpm database. One of the biggest benefits of yum is its ability to automatically handle dependencies. It means when we're installing a particular package, it will automatically install additional packages that depend on the particular package. One other benefit of yum is the use of repositories. It allows version control more easily. The yum command has several subcommands. Such as, **install** subcommand installs the specified package from any configured repository, **localinstall** subcommand install the specified package from local repository, **remove** subcommand uninstalls the specified package, **update <package_name>** subcommand updates the specified package, **update** subcommand updates all installed packages, **info** subcommand shows information about specified package, **provides** subcommand reports what package provides specified files/libraries. In addition to subcommands, **-y** option is used to automatically answer 'yes' to additional software dependencies.

Debian Packages and APT

-There are various programs that Debian has for managing its packages. In this section, we will be focusing on dpkg and APT.

- **dpkg:** Dpkg is the main package management program of Debian. Dpkg packages have *.deb* as file extension. Dpkg is used to manage packages on Debian derivatives. It has several options. Such as, **-i** option installs the specified package, **-r** option uninstalls the specified package, **-l** option lists information about the specified package, **-s** option reports whether the specified package is installed or not. Dpkg ensures that all the necessary components and dependencies are installed.
- **APT:** APT is a front-end manager for dpkg on Debian-based distributions. APT is a newer version of dpkg. It also deals with *.deb* extension packages. It has several subcommands. Such as, **install** subcommand installs the specified package, **remove** subcommand uninstalls the specified package and keeps its configuration files, **purge** subcommand uninstalls the specified package and remove its configuration files, **show** subcommand shows information about the specified package, **update** subcommand updates APT

database of available packages, **upgrade <package_name>** subcommand upgrades the specified package, **upgrade** subcommand upgrades all installed packages. Additionally, there are also **apt-get** and **apt-cache** commands and they provide more specific controls, however they are yesterday's new. One important point to notice is that *apt update* command does not install any software, only updates current one whereas *apt upgrade* command upgrades all installed software, so it will upgrade by using the installed updates (apt update).

Repositories

- **Repositories:** Repositories are *storage locations for available software packages*. Repositories are also known as *repos*. There are three types of repositories:
    1. ***Local repositories:*** Stored on the system's *local storage drive*. Version control is difficult.
    2. ***Centralized internal repositories:*** Stored on one or more systems within the internal LAN. Version control is much easier compared to local repositories due to its centralized approach.
    3. ***Vendor repositories:*** Maintained on the Internet, often by the distribution vendor. Version control is exceedingly difficult because the vendor decides what package versions are being made available.

-System administrators can also designate a specific location as a **yum repository** to be able to create internal repository. To do this, **createrepo** command is used.

- **createrepo:** This command *updates the XML files* used to reference the repository location. This repository might be on the local storage drive (local repo) or web server (centralized internal repo). After running the createrepo command, a **.repo** (repo configuration file) will be created. This configuration file will be created under **/etc/yum.repo.d/** directory.
- **.repo configuration file:** It provides additional information about the repository and is stored underneath the **/etc/yum.repos.d/** directory. Some of the .repo file components include **[repo-name]**, **name=repository_name**, **baseurl=path** (path to repo), **enabled=1** (enables the repo, 0 to disable), **gpgcheck=0** (disables GPG checking).

-Another yum configuration file is located at **/etc/yum.conf** (also contains repo). If we're using DNF, we should check **/etc/dnf/dnf.conf** file (contains repo).

-The yum command has several subcommands to view and use repositories. Some of them are: **repolist** subcommand lists all available repositories, **makecache** subcommand shows local cache information about available repositories, **clean all** subcommand clears out-of-date cache information.

-Yum also enables the synchronization of an online repository to a local storage location, and this is known as **mirroring**.

- **Mirroring:** It enables the *synchronization of an online repository to a local storage location* and reduces the way-in traffic and time to load things from the parent repository.
- **reposync:** This utility allows us to *manage mirroring process*. It will mirror an online repository to a local storage location.

-As like yum package manager, APT can be configured to access repositories. For APT, repositories can be configured inside the **/etc/apt/sources.list** file and **/etc/apt/sources.list.d/** directory.

-Like the yum repositories, APT repositories are also available on the local system (local repos), on a local network (centralized internal repos) or hosted on the internet by a vendor (vendor repos). When it comes to configuring APT package manager, we can check the configuration files under **/etc/apt/** directory which contains the configuration files for APT package manager.

Acquiring Software

-Besides installing software from repositories, we can leverage open-source software from the internet. Various sources on the internet to download open-source software include GitHub, vendor's website etc.

-In Linux, we can download software from some websites without visiting them but using CLI environments. We can use both **wget** and **curl** command to accomplish this.

- **wget:** This command is used to *access website through command line environment*. It's useful for *downloading files/software* from the internet by using the URL of the file/software. The wget command can be used in scripts to automate the process of downloading files from the internet.
- **curl:** This command is also used to *access website* and download sources from the internet in command line environment.

-There are several differences between the wget and curl commands. Let's see what they are.

- The curl command is *cross-platform* whereas wget is only command-line utility.
- The wget command can download files *recursively* whereas curl cannot download recursively.
- The wget command supports *HTTP(S)* and *FTP* only whereas curl supports more network protocols.
- The wget command is *dedicated to downloading files* whereas curl is capable of building/managing complex requests besides downloading files.

-Linux offers various utilities for managing files.

- **tar (Tape Archiver):** This utility is used for *bundling files* into a single *.tar file*. Keep in mind that there is **no real compression**. The tar utility has several options. Such as, **-c** option creates the tarball, **-x** option extracts the tarball, **-v** enables verbose mode, **-r** option appends more files to an existing tarball, **-t** test the tarball, **-f** specifies the name of the tarball in the next argument.
- **gzip:** This utility is used for *compressing files into a single .gz file*. The gzip utility can also be used to reduce the size of a tarball file by compressing it, and it would have **.tar.gz** or **.tgz** as file extension.
- **xz:** This utility is an *upgraded version of gzip* which produces a *much smaller file size*. The xz utility doesn't come installed by default in all Linux distributions.
- **bzip2:** This utility is a command-line utility that *compresses files using the Burrows-Wheeler block sorting text compression algorithm*, and *Huffman*

coding. The bzip2 utility is used to compress and decompress one single file. Such as, if we compress **.tar** file, it will have **.tar.bz2** as file extension.

- **zip:** This utility is *widely used archive file* which supports *lossless data compression*. An advantage of zip is we *don't have to create a tarball* file. We can provide multiple files to zip in order to compress them. When decompressing a **.zip** file, we need to use unzip command.

-Keep in mind that in compressing, *xz* is *better* method, *gzip* is the best for *speed*, and *bzip2* is the best *balance* between these two.

Hands-on Experience – wget, tar, gzip

-To download software from the internet using command line environment, we can use **wget** or **curl** commands. The wget command downloads over HTTP(S) or FTP protocols.

```
┌──(root㉿emre)-[/home/whiterose/Desktop]
└─# wget https://github.com/NationalSecurityAgency/ghidra/releases/download/Ghid
ra_11.1.2_build/ghidra_11.1.2_PUBLIC_20240709.zip
```

-We can bundle multiple files into a folder by using **tar** command.

```
┌──(root㉿emre)-[/home/whiterose/Desktop]
└─# tar -cvf comptia_linux.tar os.txt list.txt download.txt comptia.txt
os.txt
list.txt
download.txt
comptia.txt
```

- In this case, a tarball named '*comptia_linux.tar*' has been created and it includes *os.txt*, *list.txt*, *download.txt*, *comptia.txt* files.

-Let's compress a tarball file with a file compression utility.

```
┌──(root㉿emre)-[/home/whiterose/Desktop]
└─# gzip -9v comptia_linux.tar
comptia_linux.tar:       96.0% -- replaced with comptia_linux.tar.gz
```

- In this case, a tarball file has compressed using gzip utility.

-To extract gzip files, we need to use **gunzip** command or **gzip** command with **-d** option.

```
┌──(root㉿emre)-[/home/whiterose/Desktop]
└─# gunzip comptia_linux.tar.gz
```

Building a Software from Source Code

-In Linux, we can generally download and install applications easily through package managers. Sometimes we might not be able to download applications and run them easily. In some cases, we need to compile and install new software by ourselves. Compiling allows us to have a specific version of the software or modify the source code. Either case, we have to download the source code of the software.

- **Compiling:** Compiling software means taking the text-based source code and compiling into *binary code* for computer to understand. **Compilers** handle the *compiling process*. They translate *programming languages into machine code*.

-In Linux, there is a compiler built in by default known as **GCC** (GNU Compiler Collection). The gcc utility has the ability to support different libraries for compiling.

-When developing software, we can provide a list of all the necessary libraries for application's source code. Depending on the language used in application's source code, this is either going to be inside of a **Header File** (.h extension) or **Library File** (.a extension).

- **Libraries:** Libraries contain compiled code for common tasks. **Program libraries** are chunks of compiled code that can be used in programs to accomplish common tasks. **Shared Libraries** enable more modular programming and allow us to reduce time when compiling the software. Libraries can be accessed from */usr/lib/* (general access) or */lib/* (essential binary access) directory.

- **ldd:** This command allows user to *view shared library dependencies for an application*. It's useful when we're troubleshooting or gathering system requirements.

-Building software from its source code is called **Software Compilation**. Let's take a look at processes of it.

1. Download the file (typically compressed) and *decompress* it.
2. Go to the newly created directory.
3. Run the **./configure** command to gather system information.
4. Use the **make** command to compile the application.
5. Use **make install** command to install the binaries.

- **make:** This command *installs the application* if used without any arguments. It will be looking for the **makefile** in the current directory.
- **makefile:** This file contains *instructions used by a compiler* to build a program from source code. If we look at it in depth, this file specifies dependencies, resources, and directives for the compiler. It can efficiently rebuild the program if changes are made to source files.

Troubleshooting Software Dependency Issues

-Troubleshooting process will start out by identifying the method of installing the software, since the process might vary depending on the installation method. Let's consider using package manager.

- If we are using a package manager, we should check the configuration files in the first place.
- If we are using *APT package manager*, we should check **/etc/apt/sources.list.d/** directory or **/etc/apt.conf** file. If we are using *YUM package manager*, we should check **/etc/yum.repo.d/** directory or **/etc/yum.conf** file. If we are using *DNF package manager*, we should check **/etc/dnf/dnf.conf** file.
- Sometimes, dependencies of software were not properly installed on the system. We can verify this type of situation. In *RPM package manager*, we can check dependencies with **rpm -V <package_name>** command. In *YUM package manager*, we can use **yum info**

<package_name> to retrieve information about the package. In *APT package manager*, we can use **apt-cache show <package_name>** to retrieve information about the provided package.

- If we want to identify required dependencies in *RPM package manager*, we can use **rpm -qR <package_name>** command. In *APT package manager*, we can use **apt-cache depends <package_name>** command. In YUM package manager, we can use **yum deplist <package_name>** command.
- Also ensure that repositories contain all necessary dependency packages. Do not forget to keep repositories up to date. Run the update command to get the latest versions of the dependency packages. In *APT package manager*, run **apt update** command; in *YUM package manager*, run **yum update** command.
- Sometimes, we may encounter a broken network connection. In such cases, try to *ping repository server's hostname* or *IP address* to verify the network connectivity.

-Let's take a look at what we can do if we are having problems with manually compiled software.

- If we are trying to *manually compile a piece of software* and facing issues, it could be an issue with the compiling software itself. In such cases, we should *check the documentation* for the compiler for issues with the compiling software itself.
- To find out what libraries and dependencies are required for a particular piece of software, we should use the **ldd option <program_binary>** command.
- Verify the library file versions for program's files and versions are actually available on our system.
- When it's necessary, run the **make install** command using **sudo** to assume root privileges to run certain programs or code.

-When updating and upgrading packages, read patch documentation to understand potential issues and required actions before starting. Also, we should always have a **back out plan** and **solid backup** if something goes wrong. For instance, there

might be *compatibility problems* with other programs on the system. Additionally, we need to at once check the installation logs to see if something goes wrong.

Securing Linux Systems

Cybersecurity Best Practices

-Linux is the operating system used on most network devices and security appliances. This includes things like firewalls, Virtual Private Network concentrators, IDS, IPS, SIEM appliances etc. Securing these devices is crucial to minimize the effect of attacks.

- **Cybersecurity:** A discipline that *protects computer systems* and *digital information from unauthorized access*, *attack*, *theft*, or *data damage*. Cybersecurity relies on three specific principles, known as *CIA triad*.
    1. ***Confidentiality:*** Keeps the information and communication private and protected from unauthorized access. Confidentiality is controlled through *encryption* and *access controls*.
    2. ***Integrity:*** Keeps the organizational information accurate, error-free, and without unauthorized modifications. Integrity is controlled through *hashing*, *digital signatures*, *certificates*, and *change control*.
    3. ***Availability:*** Ensures that computer systems run continuously, and authorized users can access data. Availability is controlled through *redundancy*, *fault tolerance*, and *patching*.

-A prominent component of cybersecurity comes down to **authentication**.

- **Authentication:** Authentication enables an organization to trust that the users are who they claim to be. There are various ways to authenticate a user. Such as *PIN*, *Password*, *Passphrase* etc. In addition to using a knowledge factor, we can use **tokens**, **inheritance factor (biometrics)**.
    - ***Token:*** Token is any *unique object* (physical or digital) used to verify identity. Tokens are usually used to generate OTP (one time password).

- ***Biometrics:*** Biometrics is an authentication scheme that verifies a user's identity based on physical characteristics (fingerprint, iris, facial recognition).

-Combining different authentication methods considered as *more secure*. Putting authentication factors together is called **Multifactor Authentication**.

- **Multi-Factor Authentication (MFA):** Multi-Factor Authentication involves using more than one factor for enhanced security.
- **Remote Authentication Dial-In User Service (RADIUS):** RADIUS is an internet standard protocol that provides authentication, authorization, and accounting (AAA) services.
- **Terminal Access Controller Access-Control System (TACASS):** Provides AAA services for remote users.
- **TACASS+:** More secure and scalable than RADIUS, however only available on Cisco devices.
- **Lightweight Directory Access Protocol (LDAP):** LDAP is a TCP/IP-based directory service protocol. LDAP allows clients to authenticate to the LDAP service and the service's schema then defines the tasks that the particular client can and cannot perform once they are accessing a directory database.
- **Kerberos:** Kerberos is an authentication service based on a time-sensitive ticket-granting system. Kerberos uses symmetric key cryptography and a key distribution center (KDC) to authenticate and verify user identities. There are a few commands we need to be familiar with.
  - ***kinit:*** Initialization command that is used to authenticate Kerberos.
  - ***kpassword:*** Changes the user's Kerberos password.
  - ***klist:*** Lists the user's ticket cache.
  - ***kdestroy:*** Clears the user's ticket cache.
- **Privilege Escalation:** Privilege Escalation is the act of *exploiting vulnerability* or *misconfiguration* to gain *higher-level permissions* on a system. Poorly configured SUID and SGID permissions can lead attackers to enable privilege escalation. When an administrator uses *sudo* to perform their tasks, it's also defined as privilege escalation.

-While changing permission (SUID/GUID), consider using the *lowest permissions* needed for the task.

- **chroot Jail:** It is a way to *isolate a process* and *its children* from the rest of the system. If we want chroot jail to work, we need to ensure that the *processes don't need to be run as root*.
- **Encryption:** Encryption is a cryptographic technique that *converts data from plaintext form into coded or ciphertext*. It is one of the most fundamental cybersecurity techniques for upholding the confidentiality of data. Encryption types include *data in transit*, *data in use*, and *data at rest* encryption.

-There are several subtypes of data at rest encryption, and two of the most prominent are known as *disk/drive encryption* or *file encryption*.

- **Full Disk/Drive Encryption:** It covers *encrypting a storage drive*, *partition*, or *volume* using hardware/software utilities.
- **File Encryption:** Encrypts individual *files* and *folders* using software utilities.
- **Linux Unified Key Setup (LUKS):** LUKS is used to *encrypt storage devices in Linux systems*. LUKS uses the dm-crypt subsystem in the Linux kernel and offers compatibility with various software. LUKS standardizes the format of encrypted devices.

-Before encrypting a device, it's considered as good practice to override its contents with random data or all zeros for security. This process is known as *sanitizing the drive*.

- **shred:** This command is used to *securely wipe a storage device* by overriding it with random data or zeros.
- **cryptsetup:** This command is used as the *front-end to LUKS* and *dm-crypt*.

-Another important method in cybersecurity is *hashing*. It's really important for the integrity of the data.

- **Hashing:** Hashing is a process that transforms plaintext input into an indecipherable, fixed-length output. Hash functions make it challenging to reverse engineering the original data from the hash.

-When it comes to network security, we should implement some key things to protect data well.

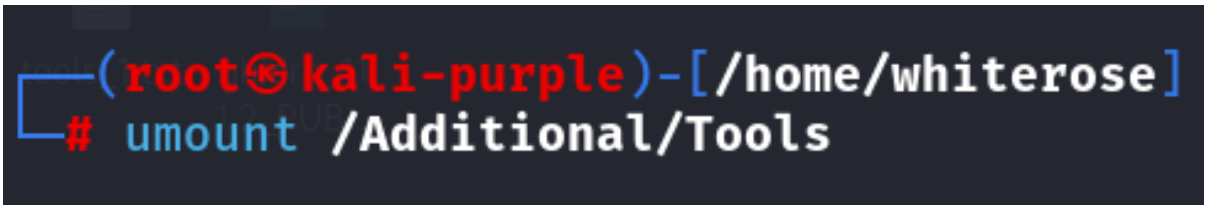
- *Enable SSL/TLS* to secure network services.
- Configure SSH to *disable the root access*.
- Change service's *default ports* to not be found easily.
- Implement *ACLs* to allow connections from trusted hosts.

-For user access security best practices, here is what we can do.

- Protect the *boot loader configuration* with a password.
- Enable *password protection* in the system's BIOS/UEFI.
- Discourage the use of *USB devices*.
- Ensure *unique User IDs* (UID).
- Establish a *public key infrastructure* (PKI) for password-less login.
- Restrict access to *cron*.
- Disable *Ctrl+Alt+Del* functionality.
- Enable the *auditd* service.
- Add a *banner message* to */etc/issue* file.
- Separate operating system data from application data into *different partitions* for enhanced availability.
- Regularly monitor the *Common Vulnerabilities and Exposures* (CVE) database.
- Harden the system by disabling or uninstalling *unused* and *insecure services*.

Encrypting a Volume – shred, cryptsetup, /etc/crypttab, /etc/fstab

-Before starting to encrypt a drive, we need to unmount it first by using **umount** command.



- In this case, the */Additional/Tools* were unmounted.

-One another thing to do before encrypting a drive is to override the disk. We can do it by using **shred** command.

```
┌──(root㉿kali-purple)-[/home/whiterose]
└─# shred -v iterations=1 /dev/mapper/extra-tools
shred: 'iterations=1': failed to open for writing: No such file or directory
shred: /dev/mapper/extra-tools: pass 1/3 (random)...
shred: /dev/mapper/extra-tools: pass 1/3 (random)...1022MiB/1.0GiB 99%
shred: /dev/mapper/extra-tools: pass 1/3 (random)...1.0GiB/1.0GiB 100%
shred: /dev/mapper/extra-tools: pass 2/3 (random)...
shred: /dev/mapper/extra-tools: pass 3/3 (random)...
```

- In this case, the */dev/mapper/extra-tools* filesystem has been overridden by *shred* command. The **-v** option enabled verbose mode, **iterations=1** argument specified the number of times to override.

-After implementing the mentioned steps, we can now encrypt the drive with a passphrase by using **cryptsetup** command and **luksFormat** subcommand.

```
┌──(root㉿kali-purple)-[/home/whiterose]
└─# cryptsetup -v luksFormat /dev/mapper/extra-tools

WARNING!
========
This will overwrite data on /dev/mapper/extra-tools irrevocably.

Are you sure? (Type 'yes' in capital letters): YES
Enter passphrase for /dev/mapper/extra-tools:
Verify passphrase:
Key slot 0 created.
Command successful.
```

- In this case, the */dev/mapper/extra-tools* filesystem has been encrypted via *cryptsetup* command. The **-v** option enabled verbose mode, **luksFormat** subcommand is used to format the device with LUKS encryption.

-Next thing to do is to create a mapping to a device name. We can do it by using **cryptsetup** command with **luksOpen** subcommand.

```
┌──(root㉿kali-purple)-[/home/whiterose]
└─# cryptsetup luksOpen /dev/mapper/extra-tools tools
Enter passphrase for /dev/mapper/extra-tools:
```

- In this case, '*tools*' is the name of the device that will be created as a mapping to the LUKS-encrypted device. After running this command, it will ask us to enter the passphrase of the drive.

-Another thing to do is to make a filesystem. We can use **mkfs** command.

```
┌──(root㉿kali-purple)-[/home/whiterose]
└─# mkfs.ext4 /dev/mapper/tools
mke2fs 1.47.1 (20-May-2024)
Creating filesystem with 258048 4k blocks and 64512 inodes
Filesystem UUID: 66940f51-c7c2-4559-9c42-d85708c0a1c3
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376

Allocating group tables: done
Writing inode tables: done
Creating journal (4096 blocks): done
Writing superblocks and filesystem accounting information: done
```

- In this case, the file system has been set as *ext4*.

-Now, we need to mount it by using **mount** command.

```
┌──(root㉿kali-purple)-[/home/whiterose]
└─# mount /dev/mapper/tools /Additional/Tools
```

-Let's add this information to the **crypttab** file. The crypttab only deals with *encrypted volumes* (similar to fstab). So, it's the place where Linux look while booting to see if there is any encrypted volume.

```
┌──(root㉿kali-purple)-[/home/whiterose]
└─# bash -c "echo tools /dev/mapper/extra-tools none >> /etc/crypttab" ; cat /etc/crypttab
# <target name> <source device>         <key file>      <options>
tools /dev/mapper/extra-tools none
```

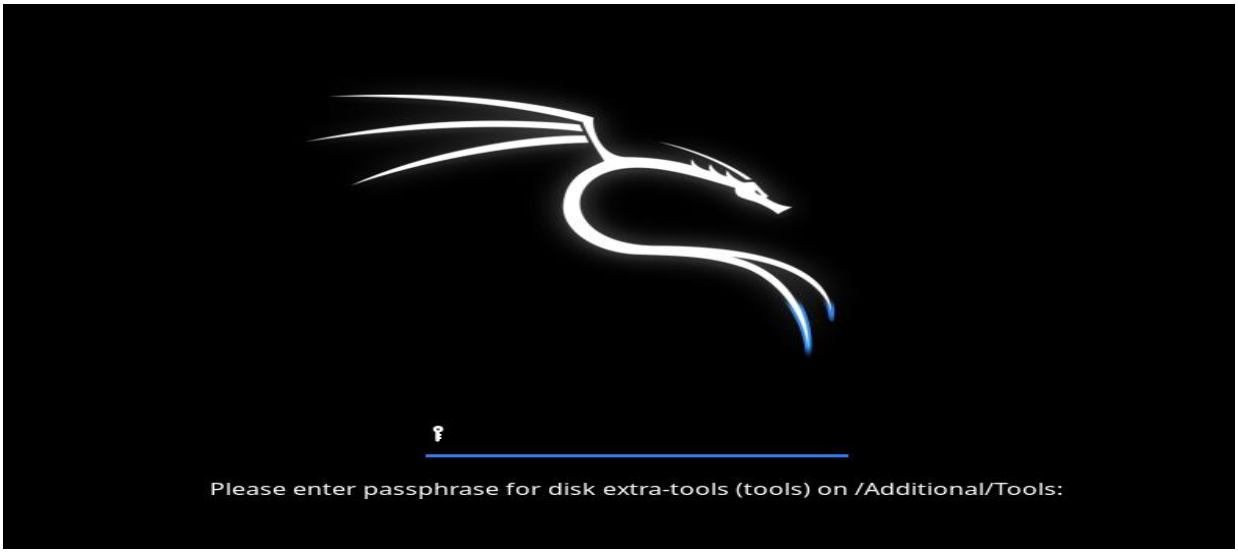
- In this case, there are two commands being executed. First, **bash -c** provides to execute the commands inside of the double quotes(" ") to run in a Bash shell. Inside of the double quotes, *encrypted device information* was written into the /etc/crypttab file. And the *semicolon* (;) provides to run another command which is **cat /etc/crypttab**.

-Now, let's edit the **/etc/fstab** file.

```
/dev/mapper/tools /Additional/Tools ext4 nofail 0 0
```

- In this case, the location of device has been replaced with the one under the **/dev/mapper** (/dev/mapper/tools). Also, the defaults argument has been replaced with **nofail** and it means mounting the encrypted volume after it has been unlocked.

-Once we reboot after implementing the steps above, Linux will prompt to enter the passphrase of the disk.



Identity and Access Management

-One of the major dimensions of cybersecurity is IAM (Identity and Access Management). IAM is considered a crucial service.

- **Identity and Access Management (IAM):** IAM is a security process that provides *identity*, *authentication*, and *authorization mechanisms* for users, end-devices to work with organizational assets like networks, applications etc.
- **Public Key Authentication:** Public key authentication is used for interactive and automated connections between different servers or between users and servers. Public key authentication provides *cryptographic strength* compared to a regular password. With SSH, public key authentication is used to improve security, also offers usability benefits, provides automated passwordless login.

-When configuring key-based authentication within the SSH, there are lots of files we might interact with. These files are located inside of the **~/.ssh/** directory. Let's look at these files.

- ***~/.ssh/id_rsa***: Contains the user's private key.
- ***~/.ssh/id_rsa.pub:*** Contains the user's public key.
- ***~/.ssh/authorized_keys:*** Lists the public keys that server accepts.
- ***~/.ssh/known_hosts:*** Contains the lists of the public keys that the client accepts.
- ***~/.ssh/config:*** Configures SSH connection settings.

-There are several commands available to work with these SSH keys.

- **ssh-keygen:** This command *generates* public/private key pair.
- **ssh-copy-id:** This command *appends* user's public keys to remote server's *~/.ssh/authorized_keys* file.
- **ssh-add:** This command adds key identities to the SSH key agent.

-Another important file is */etc/ssh/sshd_config* file.

- **/etc/ssh/sshd_config file:** This file is used to configure an SSH server on the Linux system. Let's look at the settings that we can configure here.
  - ***PasswordAuthentication:*** It is used to enable or disable password-based authentication.
  - ***PubKeyAuthentication:*** It is used to enable or disable public key-based authentication.
  - ***Hostkey:*** It is used to reference the locations of the server's private keys.
  - ***UsePAM:*** Enables or disables support for Pluggable Authentication Modules (PAM).
  - ***Port:*** It is used to change the port number to bind the SSH service (22 by default).
  - ***SyslogFacility:*** It is used to change the logging level of SSH events.
  - ***ChrootDirectory:*** It is used to reference a chroot jail path for a user.
  - ***AllowUsers/AllowGroups:*** Used to enable user-specific access by allowing the specified users or groups access over SSH.
  - ***DenyUsers/DenyGroups:*** Used to restrict the specified users/groups from accessing the server over SSH.
  - ***PermitRootLogin:*** Used to enable or disable the ability for the root user to log in over SSH.

-Additionally, we can also allow or deny connections from specific hosts to SSH server by using *TCP Wrapper*.

- **TCP Wrapper:** Checks the *allowed* and *denied hosts* before permitting the host to connect with the SSH service. We can specify the allowed hosts in **/etc/hosts.allow** file; denied hosts in **/etc/hosts.deny** file.
- **Pluggable Authentication Module (PAM):** PAM is centralized authentication framework in Linux environments. Configuration files of PAM located inside of the **/etc/pam.d/** directory. Each PAM-aware services/applications have its own file inside that directory. Each of these files include directives and these directives are formatted as four different things:

1. **<module interface>:** Defines the functions of the authentication and authorization process contained within a module. There are four module interfaces in PAM.
    - ***Account Module:*** Checks user accessibility.
    - ***Auth:*** Verifies passwords and sets credentials (Kerberos tickets).
    - ***Password:*** Changes and verifies passwords.
    - ***Session:*** Configures and manages user sessions.
2. **<control flag>:** Indicates what should be done upon a success or failure of the module. There are four predefined control flags.
    - ***Optional:*** Module result is ignored.
    - ***Required:*** Module result must be successful for authentication to continue.
    - ***Requisite:*** Notifies the user of the first failed required/requisite module.
    - ***Sufficient:*** Module result is ignored upon failure.
3. **<module name>:** Finds the module that the directive is going to apply to.
4. **<module arguments>:** Additional options that can pass into the module.

-There are also two PAM modules we should be aware of that can actually trigger a temporary user lockout if multiple authentication attempts are attempted and failed. These are known as **pam_tally2** and **pam_faillock**. The pam_faillock is recommended to use because it is a newer version of pam_tally2. We should place *user lockout directives* in **/etc/pam.d/password-auth** and **/etc/pam.d/system-auth**. We can unlock a user's account if they've logged in wrong too many times by using **pam_tally2 -r -u <username>** command.

-We can also configure PAM to use LDAP by leveraging the **pam_ldap** module. It's useful when we want to integrate Linux system in a Windows-based domain environment. We can add this directive into the /etc/pam.d/common-file.

-Password policies can be configured using PAM directives inside of the /etc/pam.d/ directory.

-To *prevent root access* over SSH, we can configure **/etc/securetty** file.

- **/etc/securetty file:** Determines the controlling terminals the root user has access to. We can accomplish to prevent root access by modifying **PermitRootLogin** variable.

-In previous sections, we covered PKI (Public Key Infrastructure) a little. Now, let's dive deeper into it.

- **Public Key Infrastructure (PKI):** PKI comprises *digital certificates*, *CAs* (certificate authorities), and *cryptographic components*. PKI can be publicly available or maintained privately by an organization. There are many cryptographic components that make up the PKI system. Some of the important ones are *Digital signature*, *Digital certificate*, *Certificate Authority*, *Certificate signing requests*.
  - ***Digital Signature:*** Encrypted message digest with a user's private key.
  - ***Digital Certificate:*** Electronic document that associates credentials with a public key.
  - ***Certificate Authority:*** Issues digital certificates for entities and maintains the associated private/public key pair.
  - ***Certificate Signing Request (CSR):*** Message sent to the certificate authority in which an entity applies for a certificate.
- **OpenSSL:** OpenSSL is an open-source implementation of the *SSL/TLS* protocol for securing data in transit using cryptography. OpenSSL is one of the most common tools for generating and managing components of a PKI.

-In order to *authenticate clients* and *encrypt data in transit*, we leverage the use of **VPN**s. VPNs use many different types of tunneling protocols. One of the most commonly used for *site-to-site connection* is known as **IPSec**.

- **IPSec (Internet Protocol Security):** IPSec is used to secure data travelling across the network/internet (data in transit) and also used in site-to-site and remote access VPNs. IPSec has two primary modes.
  1. ***Transport Mode:*** Packet contents are encrypted, whereas the header is not (Typically used for remote access VPNs).

2. ***Tunnel Mode:*** Both the packet contents and header are encrypted (Typically used for site-to-site VPNs).

-One of the most popular utilities for implementing IPSec tunnels is known as *StrongSwan*.

-SSL/TLS is another method we can use to create a VPN connection.

- **SSL/TLS:** By using SSL or TLS, we can handle *authentication* and *encryption* using the *SSL or TLS tunnel*. SSL and TLS are considered as *Application-Layer Protocol(Layer 7)* in the *OSI Model*.
- **OpenVPN:** OpenVPN is an open-source system that creates a private and secure tunnel between networks. OpenVPN supports *password-based*, *certificate-based*, and *smart card-based authentication* mechanisms for clients.
- **Datagram Transport Layer Security (DTLS):** DTLS implements *SSL* or *TLS over datagrams*, which indicates that we're using *UDP* instead of TCP.

-When troubleshooting an unsuccessful remote user authentication, we should make sure of these steps:

- Users must have set up proper credentials and transmit them to SSH or VPN.
- Check if user remote connection attempts are triggering a policy violation.
- Sign on with the local account, if remote access issue still exists.
- Ensure the user identities are correctly configured to see if it's a local issue.

Configuring SSH – ssh-keygen, ssh-copy-id, /etc/ssh/sshd_config

-We can set up public/private key pairs for SSH by using **ssh-keygen** command.

```
┌──(whiterose㉿kali-purple)-[~]
└─$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/whiterose/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/whiterose/.ssh/id_rsa
Your public key has been saved in /home/whiterose/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:9HgU6ludUBiFvLh+TiaUf5SVuWWKmcnmWqsiegGexQU whiterose@kali-purple
The key's randomart image is:
+---[RSA 3072]----+
|      E. .o=o    |
|         ..+o   o |
|     . .o.o.  + o|
|    . oo.=.+ O = |
|   . +  S.+ % o  |
|    o  ...= +    |
|       oo + +    |
|      o o+.+ .   |
|    .o . ++..    |
+----[SHA256]-----+
```

- Executing this command requires us to enter a location for keys and passphrase. In this case, -t option specified the type of key as RSA.

-After creating key pairs, we need to copy them to the remote server we'd like to connect to. We will be using **ssh-copy-id username@remote_server_ip** command.

```
┌──(whiterose㉿kali-purple)-[~]
└─$ ssh-copy-id kali@server
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/whiterose/.ssh/id_rsa.pub"
The authenticity of host 'server (192.168.50.128)' can't be established.
ED25519 key fingerprint is SHA256:zY11yo4lVmNdGVxlO5bnFqLFWnodH61tTwWrOQnuKZo.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
kali@server's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'kali@server'"
and check to make sure that only the key(s) you wanted were added.
```

- In this case, the remote server's (*server*) IP address was added into /etc/hosts file so providing server keyword worked since Linux will resolve it based on the /etc/hosts file.

-After copying key pairs with ssh-copy-id, we can connect to our remote host by using **ssh username@remote_server_ip** commands.

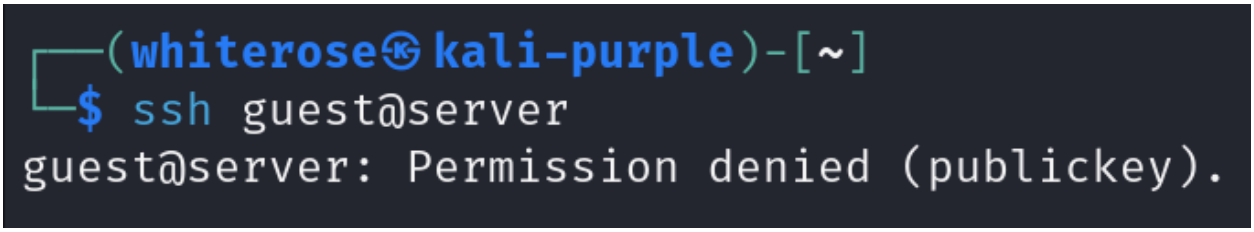
- In this case, remote hostname is *kali* and after providing the passphrase, notice the yellow circle, username in the terminal was changed to as remote host's name (kali).

-As mentioned earlier, using password for SSH is the weakest type of authentication. So, we need to disable logins with password by modifying the **/etc/ssh/sshd_config** file in remote host.

- Edit the */etc/ssh/sshd_config* file with *super user privileges*. All we need to do is to set the *PasswordAuthentication* as **no**.

-Do not forget to restart SSH daemon after editing /etc/ssh/sshd_config file. Use **systemctl restart sshd** command.

-We're not able to remotely login to a user which does not have key. Let's test it out.



SELinux and AppArmor

-Some Linux distributions provide an additional layer of security on top of the operating system to mitigate the risk of breaches. One of the best ways to stop a breach is to restrict the ability of granting or denying access to a given resource object within the system. This is where the idea of mandatory access control (MAC) comes into play.

- **Mandatory Access Control (MAC):** MAC is system-enforced access control mechanism based on subject clearance and object labels. *SELinux* and *AppArmor* are both classified as MAC, and they rely on context-based permissions.
- **Discretionary Access Control (DAC):** In DAC, each object has a list of entities that are allowed to access it, whether these are systems, files, processes etc. Person who controls that access is the actual *object owner*. They can make changes by using *chmod* or *chown* commands.
- **Context-Based Permissions:** Context-based permissions referred to permission scheme that defines various properties for a file or process. In Linux, there are two main context-based permission schemes available, known as SELinux and AppArmor.
- **SELinux:** SELinux is a security module that *provides MAC*, doesn't allow DAC, and detailed control of process labeling. It stands for Security-Enhanced Linux. SELinux was created by the *National Security Agency* (NSA). SELinux is *default* on *CentOS* and *RHEL* (RedHat Enterprise Linux). SELinux offers file system and network security, protects them against unauthorized access and data breaches. SELinux defines *three main contexts* for each file and process.
    1. ***User:*** Defines what users can access the object. Different distributions provide different users, and most common ones are *unconfined_u* (all users), *user_u* (unprivileged users), *sysadm_u* (system admin), *root* (root user).
    2. ***Role:*** Permits or denies users access to domains, resources and processes. To control this, *object_r* role applies to files and directories.

3. ***Type:*** Label portion of MAC. It groups objects together that have similar security requirements or characteristics.
4. ***Level:*** This context is considered *optional*. It describes the sensitivity level called '*multi-level security*'.

-SELinux has three different modes. *Disabled*, *enforcing*, and *permissive*. Let's explain them.

1. ***Disabled:*** SELinux is *turned off* and DAC method will be prevalent, so MAC is not going to be implemented.
2. ***Enforcing:*** SELinux security policies are *enforced*, so processes cannot violate any security policies.
3. ***Permissive:*** SELinux is enabled but the security policies are *not enforced*, so processes can bypass the security policies.

-SELinux policy is used to describe the access permissions for all of the users, programs, processes, files, and devices that are in use on the operating system. SELinux can implement two different types of policies. *Targeted* or *strict*.

1. ***Targeted:*** It's the *default* SELinux policy used in CentOS and RedHat Enterprise Linux. It only applies to *certain things* in the system.
2. ***Strict:*** *Every* system subject and object in the system is enforced to operate on MAC.

-Let's examine the commands to configure SELinux.

- **semanage:** This command is used to configure *SELinux policies*.
- **sestatus:** This command *displays* the *SELinux status*.
- **getenforce:** This command *displays* the *SELinux mode*.
- **setenforce:** This command is used to *change SELinux mode*. If we apply **setenforce 1** command, this will enable *enforcing mode*. If we apply **setenforce 0** command, this will enable *permissive mode*.
- **getsebool:** This command *displays* the *on* or *off* status of the SE Boolean values. SE Boolean values enable us to change the policy configurations at runtime without writing the policies directly.
- **setsebool:** This command is used to *change* the *on or off status* of a *SELinux Boolean value*.

- **ls -Z:** The **ls -Z** command lists contents along with each *object's security context*.
- **ps -Z:** The **ps -Z** command lists running processes along with each *process's security context*. Additionally, if we want to check one specific process, we can add the PID value at the end of this command (*ps -Z <PID>*).
- **chcon:** This command *changes the security context of a file*. It has several options. Such as **-u** option represents user, **-r** option represents role, and **-t** option represents type.
- **restorecon:** This command *restores* the *default security context*.

-Once SELinux is enforced on the system, there are going to be violation messages that are captured as part of an audit log. *Violation* occurs when an attempt to access an object or an action goes *against an existing policy*.

- **sealert:** This command makes sure that all of the alert messages for violations are being sent into the audit log. Output of this command is hard to read.
- **audit2why:** This command allows us to see violations inside of the audit logs. It's more human-readable than sealert command.
- **audit2allow:** This command is used to gather information from the logs of *denied operations* and *violations*. It's useful when we first install a system and turn SELinux on, we can see the things that are being blocked. So, we can create *allow rules* for those things. In order to do this, we can run **audit2allow -w -a** command, this command reads the audit log and display the human-readable description of the blocked activities. If we want to generate a loadable module to allow the activity to occur, we can use **audit2allow -a -M <rule_name>** command. This command will create two files under the current directory.
    1. ***<RuleName.pp>:*** This is the *policy package* file. We can load it by using **semodule -i <rulename.pp>** command.
    2. ***<RuleName.te>:*** This is the *type of enforcement* file.

-Let's talk about other context-based permission schema, AppArmor.

- **AppArmor:** AppArmor is an alternative *context-based permission scheme* and *MAC implementation* for Linux. AppArmor provides the same fundamental services as SELinux, but its approach is different. AppArmor is

much simpler than SELinux. Functionally, the main difference between AppArmor and SELinux is that AppArmor works with *file system objects* based on paths, whereas SELinux *references the inodes directly*. Each executable can have an associated AppArmor profile located under the **/etc/apparmor.d/** directory. Within a profile, we can configure two main types of rules. Capabilities and path entries.

1. ***Capabilities:*** Provide the ability for that executable to have access to some system functionality.
2. ***Path Entries:*** Enable the executable to access a specific file on the file system.

-Each profile can operate in two modes. Complain or enforce.

1. ***Complain:*** Profile violations are logged but not prevented.
2. ***Enforce:*** Profile violations are both logged and prevented.

- **Tunables:** Mechanism for tuning configuration in AppArmor without profile adjustment. Tunables are stored underneath */etc/apparmor.d/tunables/* directory. Most common tunable is in */etc/apparmor.d/tunables/home* file.

-Let's examine the commands to configure AppArmour.

- **apparmor_status:** The apparmor_status command displays the current status.
- **aa-complain:** This command places a profile in complain mode.
- **aa-enforce:** This command places a profile in enforce mode.
- **aa-disable:** This command is used to disable a profile.
- **aa-unconfined:** This command lists processes with open network sockets.

-In conclusion, AppArmor is configured to reduce the potential attack surface and provide greater in-depth defense. AppArmor can only do so much to protect against exploits in application codes.

Firewalls

-In Linux, there is a certain level of security by default. However, considering the rise of cyber-attacks, we should consider configuring firewall in our system.

- **Firewall:** Firewall is a program interface between a private network and the internet. Firewalls follow *pre-configured rules* that allow *certain traffic to pass through* from the internet to the private network. So, the firewalls can be imagined as *gateways*. There are three main generations of firewalls.
    1. ***Packet Filters:*** Packet filters are also known as *first generation firewalls* or *stateless firewalls*. Packet filters firewalls make decisions on rules that correspond to network packet attributes. These rules appear in the form of an ACL (Access Control List).
    2. ***Stateful:*** Stateful firewalls are also known as *second generation firewalls*. Stateful firewalls identify past traffic related to a packet. Stateful firewalls can view the entire conversation of a given transmission. For instance, it can detect that there was a three-way handshake. Stateful firewalls can make more informed decisions about what traffic should be denied/allowed based on what happened before.
    3. ***Application Layer Firewall:*** Application layer firewalls are also known as *third generation firewalls*. Application layer firewalls can inspect the contents of application layer traffic, such as protocols like HTTP, FTP etc. Therefore, they can make decisions based on the contents they see in those packets. Application layer firewalls can detect attempts to bypass traditional filtering and stateful inspection.
- **Stateless Firewall's ACL:** Stateless Firewall's ACL *allows* or *denies packets* based on various factors like source/destination IP address, source/destination port. Once the firewall matches traffic to a given rule in the ACL, it can perform *Accept*, *Reject*, or *Drop* the packets.

-In Linux environments, we can use various tools to manage ACLs and packet filtering. Let's explain those tools.

- **iptables:** The iptables tool uses different tables to apply *certain contexts* or *rules* and these rules are known as *chains*. It allows us to set up rules to

control incoming/outgoing traffic. We can define custom chains for more detailed control. The iptables has five different tables that could be activated by default in kernel. Filter table, NAT table, Mangle table, Raw table, Security table.

1. **Filter Table:** It is the default table which is used for typical packet filtering functionality.
2. **NAT Table:** This is used to implement Network Address Translation rules.
3. **Mangle Table:** This is used to alter the packet's TCP/IP header, and we are able to change it.
4. **Raw Table:** This is used to configure exceptions involved in connection tracking.
5. **Security Table:** This is used to mark packets with SELinux security contexts.

-By default, different rule sets within the iptables will be *lost on reboot*. In CentOS and RHEL (RedHat Enterprise Linux) we can install the **iptables-services** package and issue the **service iptables save** command. In Debian-based distributions, we can install the **iptables-persistent** package, and this utility automatically runs at boot after installation.

-Enabling logging inside of the iptables is one of the crucial things. Logging is a great method to see what is blocked/allowed. We can do this by including the log action when issuing the iptables command.

- **iptables -N <name-it-yourself>:** This command provides us to create a new chain. We can name the new chain as we'd like to.
- **iptables -I <rule> -j <chain-name>:** This command allows us to insert a rule at the beginning of a specified chain. The **-I** parameter provides us with inserting rule, -j parameter specifies a chain.

-Every iptables event is written into **/var/log/messages** or **/var/logkern.log** file.

- **Uncomplicated Firewall (UFW):** UFW makes the iptables service easier to configure. It's primarily used by home users.
- **ufw:** This command is used to *configure UFW settings*. It has several subcommands. Such as, **allow <type>** subcommand allows rules for

specified type, **logging on** subcommand turns on logging feature, **enable** subcommand enables firewall, **status** subcommand shows the status of firewall.

-We can also make complex configurations in ufw by editing a file called **/etc/default/ufw**.

- **/etc/default/ufw file:** By editing this file, we can configure high-level settings like policy defaults, and kernel module usage.
- **/etc/ufw/ directory:** This directory contains more granular configuration files. Such as, when the rules are going to be applied, customization etc.

- **Firewall Daemon (firewalld):** Firewall Daemon is used to *dynamically manage a firewall without requiring a restart* and it uses *zones* and *services* rather than rules and chains. *Firewall zones* are the rule sets that apply to network interface. There are various default zones, and each of them have different level of trust. For instance, the **drop** zone has the *lowest level of trust*. Also keep in mind that firewalld is the default firewall service in many Linux distributions.
- **firewall-cmd:** This command is used to configure firewalld by querying, adding, modifying, and deleting zones and services as desired. The firewall-cmd command allows us to permit services by name or port number. There are similar usages that we need to be familiar with. Such as, **--get-zones** option lists available firewalld zones, **--reload** option reloads zone's configuration, **--permanent** option allows us to persist changes, **--zone=<zone> --list-all** options list details of specified zone, **--zone=<zone> --change-interface=<interface>** options add specified interface to the specified zone, **--zone=<zone>--add-service=<service>** options add specified service to the specified zone, **--zone<zone> --add-port=<port-number/protocol>** options add specified port to the specified zone.
- **Netfilter:** Netfilter is a Linux kernel framework which handles packets that traverse a network interface. It can provide *packet filtering*, *NAT* and *connection tracking* therefore it works as a firewall. Netfilter (nftables) was designed as a *replacement* for *iptables* and is installed by default on Debian based distributions.

-One of the functionalities that firewalls do is IP forwarding.

- **IP Forwarding:** IP forwarding enables incoming traffic on one network interface to another. IP forwarding is useful in systems that have multiple interfaces. In case of setting up a Linux server as a firewall for the entire network, and having two NICs, we can use IP forwarding to act as a router/gateway.
- **IP Set:** IP set is stored *collection of IP* and *MAC addresses*, *network ranges*, *port numbers*, and *network interface names*.
- **ipset:** This command *creates* and *modifies IP sets*. The iptables tool provides more efficient rule matching. This command is also useful when we're troubleshooting the iptables's firewall. For instance, **test** subcommand tests whether the entry exists or not. Additionally, we can reference an IP set file from multiple servers to not configure each of them individually.

-*IANA* assigns port numbers to the most common protocols. Such as, 22 for SSH, 443 for HTTPS etc. But with uncommon apps or services, there is no standardized port number. So, we need to open these ports in the firewall. Trusted/Privileged ports include range of ports from 0 to 1023.

-If firewall blocks the traffic when it shouldn't, we better check the *firewall rule set* to ensure there are no overtly blocked ports in the system. Sometimes, the cause of the block is that we have the *protocol blocked* and not the *port blocked*. Also, ACL can be configured to only block specific source ports.

- **Intrusion Prevention System (IPS):** IPS is also a security appliance just like firewall. It monitors and evaluates a system for attack indicators and blocks traffic that it determines malicious. IPS is focused on *detecting* and *preventing malicious activities* whereas firewall is primarily concerned with controlling access to and from trusted and untrusted networks. There are many different IPS solutions available, and two common third party solutions include DenyHosts and Fail2ban.
    - **DenyHosts:** DenyHosts protects SSH servers from brute force password cracking attacks. DenyHosts monitor authentication logs to look for failed log in entries.
        - ***/etc/denyhosts.conf file:*** It's the primary configuration file for DenyHosts. We can set various options up by leveraging this

file. For instance, **ADMIN_EMAIL** defines email address to send alert, **BLOCK_SERVICE** defines services to be blocked from unauthorized users.

- **Fail2ban:** Fail2ban does not rely on one service. Instead, it monitors log files that pertain to any system service that has an authentication component. Fail2ban leverages things like iptables and netfilter to perform blocking actions and it can even update the firewall rules.
    - ***/etc/fail2ban/jail.conf file:*** It's the primary configuration file for Fail2ban. It's better to copy /etc/fail2ban/jail.local file or make a custom .conf file inside of the /etc/fail2ban/jail.d/ directory to make changes.

Configuring Firewall – firewalld-cmd

-Before starting to configure firewalld service on the system, make sure that it's enabled and running on the system. Check firewall daemon's state by using **systemctl status firewalld.service** command.

```
┌──(root㉿kali-purple)-[/home/whiterose]
└─# systemctl status firewalld.service
○ firewalld.service - firewalld - dynamic firewall daemon
     Loaded: loaded (/usr/lib/systemd/system/firewalld.service; disabled; preset: disabled)
     Active: inactive (dead)
       Docs: man:firewalld(1)

Sep 28 20:14:53 kali-purple systemd[1]: Starting firewalld.service - firewalld - dynamic firewall daemon...
Sep 28 20:14:58 kali-purple systemd[1]: Started firewalld.service - firewalld - dynamic firewall daemon.
Sep 28 20:34:15 kali-purple systemd[1]: Stopping firewalld.service - firewalld - dynamic firewall daemon...
Sep 28 20:34:15 kali-purple systemd[1]: firewalld.service: Deactivated successfully.
Sep 28 20:34:15 kali-purple systemd[1]: Stopped firewalld.service - firewalld - dynamic firewall daemon.
Sep 28 20:34:34 kali-purple systemd[1]: Starting firewalld.service - firewalld - dynamic firewall daemon...
Sep 28 20:34:34 kali-purple systemd[1]: Started firewalld.service - firewalld - dynamic firewall daemon.
Sep 28 20:34:41 kali-purple systemd[1]: Stopping firewalld.service - firewalld - dynamic firewall daemon...
Sep 28 20:34:42 kali-purple systemd[1]: firewalld.service: Deactivated successfully.
Sep 28 20:34:42 kali-purple systemd[1]: Stopped firewalld.service - firewalld - dynamic firewall daemon.

┌──(root㉿kali-purple)-[/home/whiterose]
└─# systemctl start firewalld.service

┌──(root㉿kali-purple)-[/home/whiterose]
└─# systemctl enable firewalld.service
Created symlink '/etc/systemd/system/dbus-org.fedoraproject.FirewallD1.service' → '/usr/lib/systemd/system/firewalld.service'.
Created symlink '/etc/systemd/system/multi-user.target.wants/firewalld.service' → '/usr/lib/systemd/system/firewalld.service'.
```

- In this case, it was disabled but then got enabled by using **systemctl start firewalld.service** and **systemctl enable firewalld.service** commands.

-We can display zones by using **firewall-cmd --get-zones** command. Each zones have different fields of usage.

```
┌──(root㉿kali-purple)-[/home/whiterose]
└─# firewall-cmd --get-zones
block dmz drop external home internal nm-shared public trusted work
```

-We can display active zones by using **firewall-cmd --get-active-zones** command.

```
┌──(root㉿kali-purple)-[/home/whiterose]
└─# firewall-cmd --get-active-zones
public (default)
  interfaces: eth0
```

- In this case, the public zone is enabled on eth0 interface.

-We can list currently active rules for specified zone in the firewalld service by using **firewall-cmd --zone=<zone> --list-all** command.

```
┌──(root㉿kali-purple)-[/home/whiterose]
└─# firewall-cmd --zone=public --list-all
public (default, active)
  target: default
  ingress-priority: 0
  egress-priority: 0
  icmp-block-inversion: no
  interfaces: eth0
  sources:
  services: dhcpv6-client ssh
  ports:
  protocols:
  forward: yes
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
```

- In this case, rules inside of the *public* zone was displayed. As we can see, it allows *DHCPv6* as client and *SSH* services.

-Let's start to configure firewall daemon. We can add interface to a zone by using **firewall-cmd --zone=<zone> --add-interface=<interface>** command. During this demonstration, DMZ zone will be configured. DMZ zone is used to separate internal network from external network.

```
┌──(root㉿kali-purple)-[/home/whiterose]
└─# firewall-cmd --zone=dmz --add-interface=eth0
success
```

- In this case, *eth0* interface was added to the *DMZ* zone.

-We can add service into the zone by using **firewall-cmd --zone=<zone> --add-service=<service>** command.

```
┌──(root㉿kali-purple)-[/home/whiterose]
└─# firewall-cmd --zone=dmz --add-service=ftp
success
```

- In this case, the *FTP* service was added to the *DMZ* zone.

-We can remove service from the zone by using **firewall-cmd --zone=<zone> --remove-service=<service>** command.

```
┌──(root㉿kali-purple)-[/home/whiterose]
└─# firewall-cmd --zone=dmz --remove-service=ftp
success
```

- In this case, the *FTP* service was removed from the *DMZ* zone.

-By default, the zone is set as public. To change it, we can use **firewall-cmd --set-default-zone=<zone>** command.

```
┌──(root㉿kali-purple)-[/home/whiterose]
└─# firewall-cmd --set-default-zone=dmz
success
```

- In this case, the default zone was changed to as *DMZ* zone.

-To make the changes persistent, we can use **--permanent** option at the end of the command.

```
┌──(root㉿kali-purple)-[/home/whiterose]
└─# firewall-cmd --zone=dmz --add-service=http --permanent
success
```

-To apply the changes, we need to use **firewall-cmd --reload** command.

```
┌──(root㉿kali-purple)-[/home/whiterose]
└─# firewall-cmd --reload
success
```

Logging Services

-Besides designing a secure system and implementing rules, logging is crucial to be aware of *misconfigurations* or *identifying anomalies* on the system.

- **Operating System Logs:** Operating system logs provide a wealth of *diagnostic information* about a computer. Everything from kernel events to user actions is being logged on Linux.
- **System Log:** System logs are records of *system activities* and *events*. System logs are tracked by syslog daemon.
- **Remote Logging:** Remote logging is *centralized logging server* that receives and processes syslog data.

-In most Linux distributions, system logs stored in **/var/log/** directory by default. Inside of that directory, each file and directory correspond to a different service's, application's, or operating system's log. Let's examine them.

- **/var/log/syslog file:** This file contains all types of *system events* in *Debian-based distributions*.
- **/var/log/auth.log file:** This file contains *authentication messages* in *Debian-based distributions*.
- **/var/log/messages file:** This file contains *non-critical system events* in *RedHat/CentOS distributions*.

- **/var/log/secure file:** This file contains *authentication messages* in *RedHat/CentOS distributions*.
- **/var/log/kern.log file:** This file contains *kernel messages* in *both* RedHat/CentOS and Debian-based distributions.
- **/var/log/ directory:** This directory contains *applications/services* logs.

- **Log Rotation:** Log rotation is the practice of creating *new versions of a log file*. It improves log management and analysis. Log files can be rotated according to size, time, or other criteria.
- **logrotate:** This utility is used to perform *automatic rotation of logs*. This utility can also be configured in **/etc/logrotate.d/** directory. We can define log's permissions, size etc.
- **rsyslogd:** Rsyslogd is a service which is an *enhanced version of the syslog service* in Linux. It supports TCP, SSL/TLS encryption, output to databases etc. Rsyslogd maintains *configuration files* similar to syslogd for compatibility. We can find the configuration file of rsyslogd service in /etc/rsyslog.conf file, and this file determine how to handle syslog messages.
- **syslog-ng:** It's the *new replacement* for syslogd. It offers similar functionality with its own syntax and features.

-The syslog standard is not universally supported on all platforms. For instance, Windows uses a proprietary format (Windows Event Log) to record system messages by default. So, in order to facilitate interaction between syslog and non-syslog platform, we need to use *third party agents*.

- **Third-Party Agents:** First of all, *agent* is a software program that acts on behalf of some other program or service. Third-party agents enable integration of non-syslog platforms with syslog.
- **journalctl:** This command is used to *view* and *query logs* that are created by the journal component of the systemd suite. It can display logs, filter by severity, service, and more. The journalctl settings can be configured in **/etc/systemd/journald.conf** file. The journalctl command has several options. Such as, **-n** option specifies number of lines of journal logs to display, **-o** option specifies format of the output, **-f** option displays most recent journal entries, **-p** option filters journal log output by severity, **-u**

option filters journal log output by name of service, **-b** option shows log message from specified boot ID.

- **/var/log/journal/ directory:** This directory contains systemd *journal logs*. Also, allows logs to persist after a reboot.
- **last:** This command displays the *user's history of login* and *logout events* along with the actual time and date. It also allows us to filter results, such as filtering by the users who have logged in through a specific terminal. The last command retrieves information from **/var/log/wtmp** file.
- **lastb:** This command lists *bad login attempts*.
- **lastlog:** This command lists *all users* and *last time a user logged in*. The lastlog command retrieves information from **/var/log/lastlog** file.

Backup, Restore, and Verify the Data

-In cybersecurity, having backed up systems is a real lifesaver thing in case of data loss due to cyberattack, human errors, physical damage to the IT components etc.

- **Backup Strategy:** Backup strategy is important thing in data protection which directs data backup and recovery policy actions. Backup strategy is not a thing to create one time and use all time, but it needs to be implemented every day.
- **Backup:** In simple terms, backup is copy of data that exists in another logical or physical location other than the original data itself. Process of recovering data from backup depends on the backup type used in backup plan. There are many types of backups with different purposes.
  - ***Full Backup:*** All selected files are backed up. This type of backup allows us to recover quickly.
  - ***Differential Backup:*** Focuses on the files that have changed since the last full backup. It requires less storage space than a full backup, but the recovery time is slower.
  - ***Incremental Backup:*** Only backs up the changes that have been made since the last back up whether that was a full or a differential backup. It is faster to perform this type of backup but slower to recover.

-Memorize the differences between backup types as '*Full backup* is the **most**, *Differential backup* is the **medium**, and *Incremental backup* is the **least** amount of data that's being backed up'.

- **Snapshot:** Snapshot records the *state of a storage drive at a certain point in time* and is located on the same drive.
- **Image-Based Backup:** Image-based backup saves the *state of an operating system in an image file format* (ISO).
- **Cloning:** Cloning is the process of *copying all the contents of a storage drive to another storage medium*.
- **tar (tape archiver):** This command enables the creation of data archives. The tar command can also direct its output to an available device, file or other programs using pipe. For instance, **tar -xvf** command restores the contents of a source file.
- **dar (disk archiver):** This command offers more backup and archiving functionality. The dar command is intended to replace tar. For instance, **dar -R <data> -c full.bak** command enables us to create a full backup of data, **dar -R <data> -c diff.bak -A full.bak** command enables us to create a differential backup, **-x** (extract) option enables us to recover a backup.
- **cpio:** This command is used to *copy files to and from archives*. Syntax of the cpio command depends on the mode, and it reads from standard input (STDIN). The cpio command has three operating modes. Copy-out, copy-in, and copy-pass.
  - ***Copy-out:*** Used to copy files into an archive.
  - ***Copy-in:*** Used to copy files from an archive.
  - ***Copy-pass:*** Used to copy files from one directory tree to another.
- **dd (disk duplicate):** This command *copies* and *converts files to be transferred from one type of media to another*. The dd command has various operators. First, we need to specify an input file (where data to be read from) by using **if=<file-name>** argument, then **of=<file-name>** argument to specify output file (where data to be written to). The **bs=<bytes>** argument to specify total block size to read and write in bytes, **count=<count>** argument to specify the number of blocks to be written, **status=<level>** argument to specify information to print to standard error.

-Important thing to know that everything in Linux is treated as **file**, so the files keyword may refer to USBs, hard drives etc.

- **mirrorvg:** This command is used to create copies (mirrors) of logical volume in a specified logical volume group. It can create multiple copies for redundancy.
- **mklvcopy:** This command mirrors individual logical volumes in a volume group.

-We can also create one or more mirrors of a logical volume by using the **lvcreate** command with **-m<number-of-copy>** option. For instance, imagine creating a *25 GB* mirror volume named *my-mirror*, and copy from *user-data* volume group; to do this we can use **lvcreate -L 25G -m1 -n my-mirror user-data** command.

-While backing up the data, it's worth considering where we are going to place the backups. Assume that we've placed our backup drives next to the main servers, and in case of disasters like flood, fire etc., the backup drives would have been gone. So, we need to ensure that we have a solid off-site backup.

- **Off-site:** Off-site refers to a physical location outside of the main site that stores copies of data. Usually, it's convenient to connect main site and off-site to each other via VPN and transfer data over the network or using a cloud-based off-site backup would do it as well.

-There are a lot of useful data transfer tools in Linux. Such as scp, sftp, and rsync.

- **scp (Secure Copy):** This tool is used to copy data to or from a remote host over SSH. Because of using SSH, the moving data is encrypted and being sent over the network.
- **sftp (Secure File Transfer Protocol):** This tool uses an SSH tunnel as a transportation mechanism to encrypt the data. It's useful when doing full system backups.
- **rsync:** This tool is used to copy files locally and over to remote systems. Its benefit is the efficient use of network bandwidth. It does not copy all files, instead it will only copy differences between source and destination files.

Hands-on Experience – tar, rsync

-Before backing up data, it's good to create an archive by using **tar** command.

```
┌──(root㉿kali-purple)-[~whiterose]
└─# tar -cvf customer_info.tar customer_info/*
customer_info/address_info
customer_info/credit_card_info
customer_info/user_info
```

- In this case, everything under the *customer_info* directory was added into the *customer_info.tar* archive.

-We can display the content of tar file by using **tar -tf** command.

```
┌──(root㉿kali-purple)-[~whiterose]
└─# tar -tf customer_info.tar
customer_info/address_info
customer_info/credit_card_info
customer_info/user_info
```

- In this case, the *customer_info.tar* file consists of *address_info*, *credit_card_info*, and *user_info* files.

-Using tar is not such an efficient way of copying information from one location to another that needs to be constantly updated. We can use **rsync** command so if there are any changes, only that information is being sent over rather than the full drive.

```
┌──(root㉿kali-purple)-[~whiterose]
└─# rsync -av sales_info /Additional/Tools/backup
sending incremental file list
sales_info/
sales_info/AccountInfo.txt
sales_info/customer_info.tar
sales_info/sales_report_april.jpg
sales_info/sales_report_february.jpg
sales_info/sales_report_january.jpg
sales_info/sales_report_march.jpeg
sales_info/sales_report_may

sent 852,200 bytes  received 153 bytes  1,704,706.00 bytes/sec
total size is 851,356  speedup is 1.00
```

- In this case, **-av** option allows archive and verbose mode. The *sales_info* folder was sent to */Additional/Tools/backup* directory. We can see detailed information on the output.

-Now, let's append a file into the sales_info directory and send it to /Additional/Tools/backup directory with rsync to see if only the changes or the full drive are sent.

```
┌──(root㉿kali-purple)-[~whiterose]
└─# echo "Sales Report - August 2024" >> ./sales_info/sales_report_august.txt

┌──(root㉿kali-purple)-[~whiterose]
└─# rsync -av sales_info /Additional/Tools/backup
sending incremental file list
sales_info/
sales_info/sales_report_august.txt

sent 451 bytes  received 39 bytes  980.00 bytes/sec
total size is 851,383  speedup is 1,737.52
```

- In this case, *sales_report_august.txt* file was added into *sales_info* folder and it's synced by using **rsync**. As result, we can see that *only the change* was being sent over.

-Last important thing is to check the integrity of backups. To check the integrity, we can leverage **hashes**. It can be done via **sha256sum** command.

```
┌──(root㉿kali-purple)-[~whiterose]
└─# sha256sum sales_info/* > hash-sales_info.txt

┌──(root㉿kali-purple)-[~whiterose]
└─# cat hash-sales_info.txt
786b003afeacfee69c74d72e06d2dd60be6faa846b9499cdd535fd4692fa6539  sales_info/AccountInfo.txt
943210dd817e5bca9f04eb37dbc34876e314f68d44616537055376 66f163bd0d  sales_info/customer_info.tar
4d187c968e2080ec1f726be076e73e68434992518992cc1b79dbea983fce05e1  sales_info/sales_report_april.jpg
0bea8a33354f7b19622eb59c5e547ec262ed67ec88e8959697419bdf3a5119c9  sales_info/sales_report_august.txt
3cfee494d768e3451480905d940f33392bc2891d9322c483cd181834ba953718  sales_info/sales_report_february.jpg
8c7bbd8d73b6f306b05f9ac48d06e05ee716b747bef1015f77e4dc1a2208207f  sales_info/sales_report_january.jpg
4af588bd3337faf5dbb8a0e4c444c2d176ba8e7c7c54ca6bd69f6f340f074aae  sales_info/sales_report_march.jpeg
1df5bd1587679b1a4e7d1cbfafbf50c21ba9928ce179012f715f86349fb863b9  sales_info/sales_report_may
```

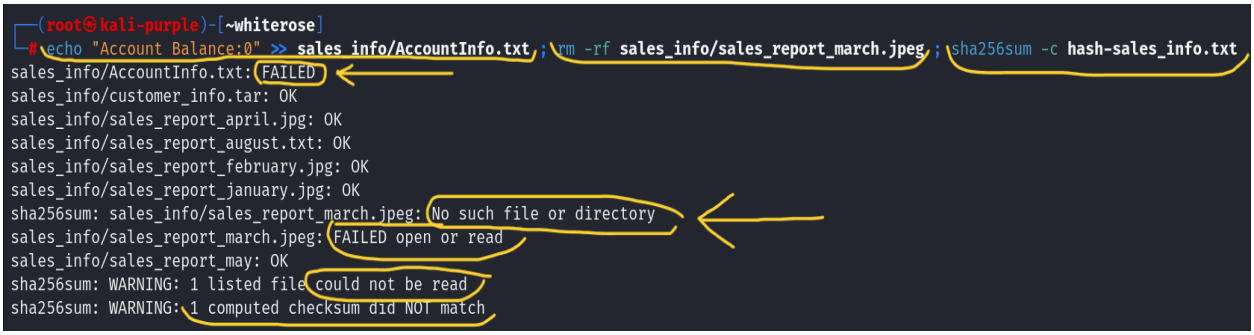
- In this case, *SHA-256* hashes were created for everything underneath *sales_info* folder.

-We can compare the hashes with the files/folders by using **sha256sum -c <hashes>** command.

```
┌──(root㉿kali-purple)-[~whiterose]
└─# sha256sum -c hash-sales_info.txt
sales_info/AccountInfo.txt: OK
sales_info/customer_info.tar: OK
sales_info/sales_report_april.jpg: OK
sales_info/sales_report_august.txt: OK
sales_info/sales_report_february.jpg: OK
sales_info/sales_report_january.jpg: OK
sales_info/sales_report_march.jpeg: OK
sales_info/sales_report_may: OK
```

- In this case, process of checking hashes went through successfully.

-Let's make modifications on the files in order to see if the checking hashes will fail or not.



- In this case, a text file was modified, and a file was removed. So, these changes caused the checking hashes to fail.

Bash Scripting

Bash Shell Environment

-Beside interacting with computer via GUI, we can also interact with computer via shell environment.

- **Shell Environment:** Shell environment is a mechanism by which Bash maintains settings and other behavioral details.
- **Shell Spawning:** It's the process of creating a *new session*, and these sessions can become *child processes*. For instance, the shell spawns a child process when the user enters a command. This child process becomes the new process and can also create other sub-processes or child processes itself. This can result in multiple generations of processes in *nested tree architecture*. Each process is going to call upon the shell environment and pass it details onto the next generation until it *reaches the kernel* and *processor*.
- **Script:** Any computer program that automates the execution of tasks.
- **Variable:** Variable refers to things where the values can change from time to time. Variables can be either *shell variables* or *environment variables*. *Shell variables* do not pass values to child processes, while *environment variables* do. We can set variable in **VARIABLE=VALUE** format in bash scripting.

For instance, **COUNT=5**. We can reference variables by using **${VARIABLE-NAME}** format. For instance, **${COUNT}**.

- ***Environment Variable:*** Variable that is *inherited from parent shell processes* and passed to the child processes. These variables consist of a *name*, usually written in *upper case letters*, and a *value*. Some of the default environmental variables include **HOSTNAME={hostname}**, **XDG_CURRENT_DESKTOP={desktop-environment}**, **PATH={user-path}**. Environment variables can also be used to configure localization options by editing */etc/locale.conf* file. These localization options can be configured using environment variables like **LC_***, **LANG**, **LC_ALL**, and **TZ**.

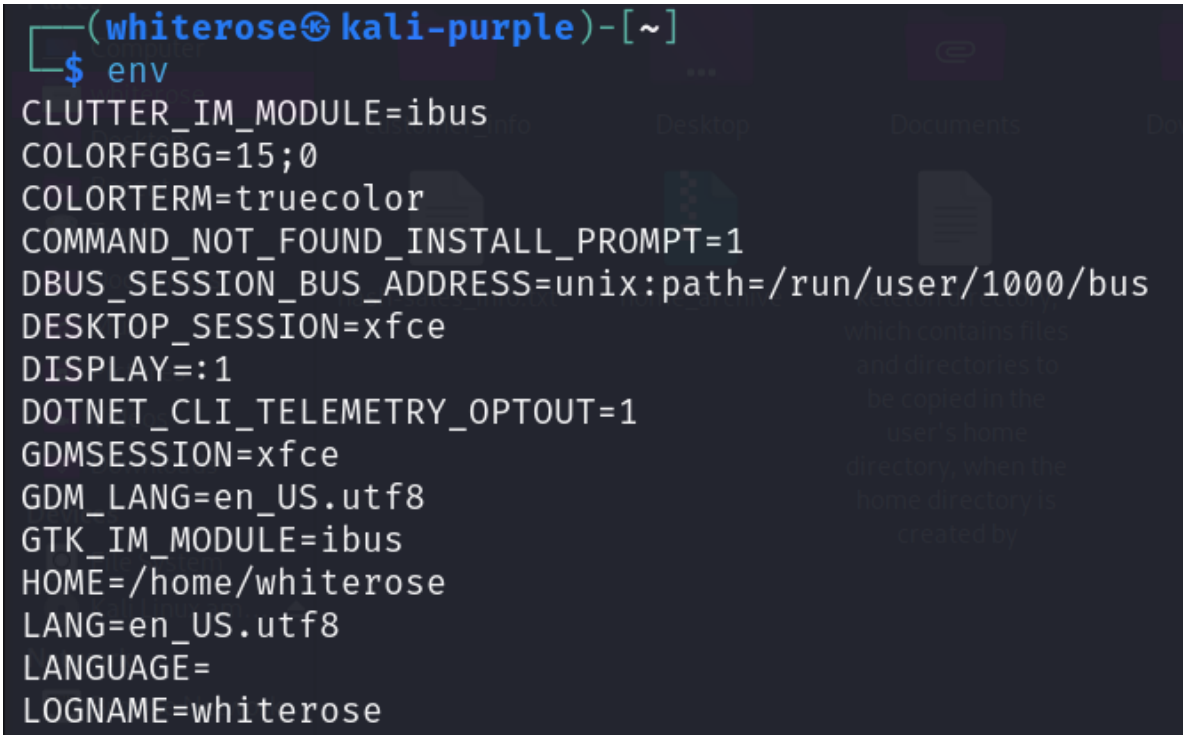
- **export:** This command is used to *change a shell variable to an environment variable*. Such as, **export IP=192.168.1.55** command will create an environment variable called *IP* and its value as *192.168.1.55*.
- **env:** This command is used to run a command with *modified environment variables*.
- **set:** This command prints *all shell variables*, *environment variables*, and *shell functions* when used without arguments. Also, we can change their behavior by using its options.
- **Search Path:** It's a sequence of *various directory paths* that's going to be used by shell *to locate files*. These paths can be assigned to the path environment variable (**PATH=path**) and this variable has a list of all directory names separated by colons.
- **HISTFILESIZE:** This variable sets the maximum number of lines in command history file.
- **alias:** This command is used to customize the shell environment by generating command-line aliases.
- **time:** This command is used to gather information about how long it takes to execute the provided command. Also, it includes additional statistics about the *real time* that elapsed between invocation and termination, *user CPU time*, and *system CPU time*.

-When encountering an issue related to variables, we can follow these steps to troubleshoot.

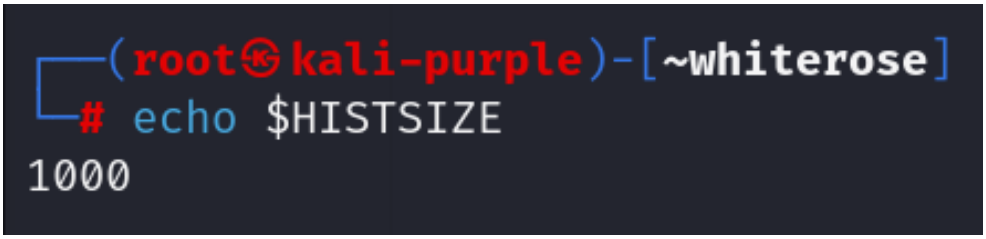
- When adding an alias, check the syntax.
- When executing scripts, add their location to the PATH variable.
- Use the export command to set a variable for all shell child processes.
- Configure environment variables in the ~/.bash_profile file.
- Ensure values are set for any environmental variables that a software package has a dependency on.

Hands-on Experience – env, export, bash_profile

-We can list all of the environment variables by using **env** command.



-We can print the value of an environment variable by using **echo $<variable>** command.



-We can change the value of an environment variable by using **export** command.

```
┌──(root㉿kali-purple)-[~whiterose]
└─# export HISTSIZE=2000

┌──(root㉿kali-purple)-[~whiterose]
└─# echo $HISTSIZE
2000
```

- In this case, the HISTSIZE variable has changed from the value of 1000 to as 2000.

-As we can remember, the **PATH** variable display locations that Linux will look to see how to execute a command. In order to change the PATH variable, we can modify the **.bash_profile** file.

```
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/.local/bin:$HOME/bin

export PATH=$PATH:/my_binary
```

- In this case, *my_binary* directory was added to the PATH. Do not forget to use **source** command to update the PATH (**source .bash_profile**).

```
┌──(root㉿kali)-[/home/kali]
└─# echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/local/games:/usr/games:/root/.local/bin:/root/bin:/my_binary
```

- As we can see, *my_binary* directory has been added successfully.

Scripting and Programming

-Bash scripting would help us to get things done quicker and easier. We can leverage bash when writing automation to our daily tasks. Keep in mind that in Linux shell environment, bash is not the only language we could use.

-Bash scripting supports modern programming elements like loops, conditional statements. Syntax of Bash scripting is similar to CLI.

-In our bash script file, we need to add a line called **Shebang Line**.

- **#!/bin/bash:** This line instructs the operating system to use the bash shell interpreter. We need to add this line to the first line of our bash script file.
- **Assigning Variable:** Inside of our script, assigning variable refers to a specific piece of information with a given name. This includes strings, integers, floats etc. In bash scripting, *all bash variables* are treated as **strings**. When assigning value to a variable in bash scripting, make sure that there are *no spaces before or after the equal sign* (my_var='CompTIA Linux+' is correct format).

-We can call the previously assigned variables in our bash script. This referencing is known as **Substitution or Parameter Expansion**.

- **Substitution or Parameter Expansion:** It's the act of referencing/retrieving the value of a variable.

-Note that when assigning a variable, we don't use a dollar sign, but when pulling information from that variable, we use dollar sign.

-In addition to store and retrieving data from a variable, we can perform various operations with variables. For instance, we might need to evaluate the condition of a variable and what's stored inside of it. This is called *comparison*, and it's done by using **operators**.

- **Operators:** They are objects that can evaluate expressions in different ways. Bash supports various types of operators. *Arithmetic*, *Comparison*, *Logical*, and *String operators*.
    - ***Arithmetic Operators:*** These operators include things like *addition*, *subtraction*, *multiplication*, *division*, and other operations.

- ***Comparison Operators:*** These operators are used to check if things are *less than*, *greater than*, or *equal*.
- ***Logical Operators:*** These operators connect multiple values (*AND*, *OR*, and *NOT*).
- ***String Operators:*** These operators are used in operations that manipulate strings in different ways including *concatenate strings*, *returning a specific character in a string* (slicing), or *verifying if special characters exist inside of a string*.

➢ **String Literal:** A string literal is any fixed value that represents a string of text within the source code. They are enclosed with either single or double quotation marks. It's good practice to put quotes around strings being assigned to a variable.

-In programming languages, including bash, there are characters that have special meanings/usages. These types of characters are called **reserved characters**. Such as *escape character (\)*. **Escape Character** is used to remove special meaning so that any character can be used literally. Escape character in bash is a *single backslash* (\). For instance, **echo hello #mate** command will print **hello** whereas **echo hello \#mate** command prints **hello #mate**. Keep in mind that **echo 'hello #mate'** command also prints **hello #mate**.

➢ **Array:** Array is collections of values. Arrays enable us to store multiple values in a single variable. Elements of array can be easily updated because it's assigned to a single thing instead of multiple. Arrays are ordered based on indices, so usually starting from zero. Compound assignment in bash arrays uses *parentheses with a value separated by space*. For instance, **my_os=("Kali" "Linux" "2024.2")** array consists of three elements and **my_os[0]** corresponds to Kali element.

➢ **Function:** Functions are blocks of code that can be reused to perform a specific task. It's useful when we want to perform same things in the same script over and over again. In bash, there are two ways to write a function.
1. ***First method:*** function my_function_name { script … }
2. ***Second method:*** my_function_name() { script … }

➢ **Comment Line:** Comment line is used to explain the code and it is not executed by the system. In bash, we can create a comment line by using **hashtag** (#).

- **Metacharacters:** Metacharacters are special characters that the bash shell will interpret in a certain way. These characters have to be escaped or enclosed in quotes if we want them to be interpreted literally. Let's take a look at the metacharacters.
  - **>:** Used for *output redirection*, overwrites the output in file (pwd > myfile.txt eg.).
  - **>>:** Used for output redirection too, appends the output into the file (whoami >> users.txt).
  - **<:** Used for *input redirection*, it gets the input for the command to the left from the file listed to the right of the symbol (sort < myfile.txt eg.).
  - **<<:** Used for input redirection too, it creates a new input stream within the script.
  - **|:** Used for *piping* (pipe content from one command to another).
  - **":** Used for *defining weak string literals*.
  - **':** Used for *defining strong string literals*.
  - **`:** Used for *breaking out a string literal*.
  - **\:** Used for *escaping characters*.
  - **=:** Used for *variable assignment*.
  - **$:** Used for *variable substitution* and other types of shell expansion.
  - **#:** Used for *commenting*.
  - **||:** Used for logical **OR** operations.
  - **&&:** Used for logical **AND** operations.
  - ***:** Used for *wildcard matching*.
  - **?:** Used for *wildcard matching a single character*.
  - **[]:** Used for *wildcard matching any characters between brackets*.
  - **{}:** Used for *parameter substitution* and *arrays*.
  - **():** Used for *grouping commands*.
  - **&:** Used for running a *process in background*.
  - **; (semicolon)**: Used for *separating multiple commands* on the same line.
  - **!**: Used for *referencing command history*.

- **Exit Code/Exit Status:** Exit Code/Status is known as programs that can pass a value to a parent process while terminating. In Linux, **status code of zero** indicates that process executed *successfully*, **status code of one or higher** indicates that there is an *error* encountered.

-As we have covered in previous sections, we can make redirection or piping of status code as STDOUT/STDERR/STDIN in our command line interface.

-Bash interprets commands as splitting things up into tokens (words).

- **Shell Expansion:** It's the process by which the shell identifies special tokens.
- **Variable Substitution:** Variable substitution is a type of shell expansion by which the shell identifies the dollar ($) sign special character and expands into its actual value.

-Bash performs its expressions in a defined order.

1. Brace Expansion
2. Tilde Expansion
3. Parameter and Variable Expansion
4. Arithmetic Expansion
5. Command Substitution
6. Word Splitting
7. Filename Expansion

- **Globbing:** Globbing pattern is used for matching or expanding specific types of patterns. There are three most commonly used wild card characters when creating a globbing pattern. Asterisk (*), question mark (?), and square brackets ([]).
  - ***Asterisk (*):*** Used to match any number of characters. For instance, **mv ~/Desktop/*.txt ~/Documents** command would move every file that end with .txt to ~/Documents, so it means that anything .txt match the criteria.
  - ***Question Mark (?):*** Used to match a single character. For instance, **mv ~/Desktop/?.txt ~/Documents** command would move every file that is named as a *single character* and end with .txt to ~/Documents, so if there are files named **emre.txt** and **e.txt**, it will **only** move **e.txt**.

- ***Square Brackets ([]):*** Used to match any of the characters listed. For instance, **mv ~/Desktop/[em] ~/Documents** command would move every file that has **combination of e and m** letters to ~/Documents.

- **Positional Parameter:** Positional parameter is a variable within a shell script that is assigned to an argument when the script is invoked.
- **exec:** This command replaces the bash with the command to be executed. It does not create a new process. It's useful if we want to prevent the user from returning to a parent process inside our script, when there's an error encountered.
- **source:** This command is used to execute another command within the current shell process. It's useful when we'd like to stay within our current shell when we're executing a script.

-After creating a bash script file, we need to give it executable permission by using chmod command. Otherwise, it can't run. Also, we need to make sure that we have write and execute bits set on the directory that contains the bash script file.

Bash Fundamentals

- **Bash:** Bash is a command line scripting language used for the command shell inside Unix-like systems like Linux and macOS. Bash is useful when automating tasks. A downside of bash is that it's **not** an object-oriented programming (OOP) language.

-Bash script files must contain with #!/bin/bash line to tell the operating system that it should interpret in bash.

-When assigning a variable inside of our bash file, we can declare it as follows:

- my_variable=value
- My_variable=value
- ***declare [option] <Variable-name> = value:*** Besides assigning variables like above, we can also use this syntax. Also notice that variable name must start with capital letter. Examples:
  - **declare** -i Age = 21 => **-i** option defines integer.
  - **declare** -r Pi = 3.14 => **-r** option makes the variable constant.

-When creating arrays in bash, we can create them as follows:

- array_name = (value1, value2, value3, value4)
- array_name[position]
- **array_name[0]** => value1
- **array_name[1]** => value2
- **array_name[2]** => value3
- **array_name[3]** => value4

-We also have Named and Associative Arrays, we can assign by using **declare** keyword.

- **declare -A Array_name**
- **Array_name[key] = "value1"**
- **Array_name[key2] = "value2"**
- **${Array_name[key]}**
- **${Array_name[key2]}**

-In order to be able to program in bash file, we need to create a file with **.sh** extension and give it executable permission by using **chmod** command.

```
┌──(root㉿kali-purple)-[/home/whiterose/Desktop]
└─# touch my_bash.sh

┌──(root㉿kali-purple)-[/home/whiterose/Desktop]
└─# chmod +x my_bash.sh
```

-Let's talk about doing comparison in bash. Flow controls are going to be based on doing some kind of comparison. In arithmetic comparison, instead of using symbols (=, >, < etc.), we actually use text. But remember that we can use arithmetic symbols as well.

- '**if ["$a" -eq "$b"]**': It checks if *a* equals *b* and returns 0 or 1 as exit status.
- '**if ["$a" -ne "$b"]**': It checks if *a* not equal to *b*.
- '**if ["$a" -gt "$b"]**': It checks if *a* is greater than *b*.
- '**if ["$a" -ge "$b"]**': It checks if *a* is greater than or equal to *b*.
- '**if ["$a" -lt "$b"]**':It checks if *a* is less than *b*.
- '**if ["$a" -le "$b"]**':It checks if *a* is less than or equal to *b*.

-We can compare strings in bash as well. It's called string comparison. Notice that we will be using arithmetic symbols instead of using text-based comparison (-eq, -lt etc.).

- '**if ["$a" = "$b"]**' or '**if ["$a" == "$b"]**': Checking if the two words *equal to each other*. If one of them had capital letter and other had lower case letter, they're not going to be equal, and we'd get **one** as return as exit status.
- '**if ["$a" != "$b"]**': It checks if the strings not equal to each other.
- '**if ["$a" \< "$b"]**' or '**if [["$a" < "$b"]]**': Checking if the *a* string is *less than* (*in ASCII alphabetical order*) the *b string*. If we don't want to use escape character (\), we need to use double square brackets ([[ ]]).
- '**if ["$a" \> "$b"]**' or '**if [["$a" > "$b"]]**': Checking if the a string is *greater than* (*in ASCII alphabetical order*) the *b* string.

-Bash has an opportunity to compare logical expressions. This is called logical comparison. It can be done by using **if statement**.

```
if [condition]
   then
      # code
elif [condition]
   then
      # code
else
      # code
fi
```

→The if statement in bash must be closed with **fi** keyword at the end.
→Notice that there is a keyword named **then** after *if* and *elif* statements.

-Let's talk about loops in bash. Loops include **for**, **while**, and **until** commands in bash.

- ***For Loop:*** Performs a set of commands for each item in a list.

```
for variable in list_of_items
do
   # code
done
```

→Notice the **do** and **done** keywords. We need to write our commands between do and done keywords.

- ***While Loop:*** Performs a set of commands while a test is true.

```
while condition
do
   # code
done
```

→Notice the **do** and **done** keywords, same as the for loop's.

- ***Until Loop:*** Performs a set of commands until a test is true.

```
until condition
do
   # code
done
```

→Notice the **do** and **done** keywords, same as the for and while loop's.

➢ **<u>String Operations:</u>** String operations are the commands used to manipulate data in string format things like words. Let's try to make sense of it on example.

```
#!/bin/bash

my_var="Hello World!"
echo $my_var
echo ${my_var:2:5}
```

→The **echo $my_var** will print *Hello World*! which is the *full version* of my_var variable.

→But **echo ${my_var:2:5}** command will print starting from 2nd element and count 5 characters. It's **llo W** (Space is considered as a character).

```
┌──(root㉿kali-purple)-[/home/whiterose/Desktop]
└─# ./bash.sh
Hello World!
llo W
```

-Let's talk about inputting and outputting data in bash. We can get input from user by using **read** command. The echo is not the only way of outputting. We can also use **printf** command.

```
#!/bin/bash

read -p "Distro name: " distro_name
printf "Your OS is $distro_name Linux"
```

→If the **read** command is used with **-p** option as in the example above, we can display a message before getting the input from the user. The **printf** command works similar to echo command.

```
┌──(root㉿kali-purple)-[/home/whiterose/Desktop]
└─# ./bash.sh
Distro name: Kali
Your OS is Kali Linux
```

-In bash scripting, we can also read or write data into files. To do this, we need to create a variable which will act as that specified file. Let's see it on example.

```
#!/bin/bash

laptop_brands=$(<list.txt)
printf "$laptop_brands"
```

→When reading from a file, we need to use less than (<) symbol before the file name. Keep in mind that, < is for **reading data**, > is for **writing data**.

```
┌──(root㉿kali-purple)-[/home/whiterose/Desktop]
└─# cat list.txt
BRAND           CPU             GPU             RAM
Lenovo          Ryzen 9         RTX 4090        32 GB
MacBook         M3               -              8 GB
ASUS            Intel i9        RTX 4070        16 GB

┌──(root㉿kali-purple)-[/home/whiterose/Desktop]
└─# ./bash.sh
BRAND           CPU             GPU             RAM
Lenovo          Ryzen 9         RTX 4090        32 GB
MacBook         M3               -              8 GB
ASUS            Intel i9        RTX 4070        16 GB
```

→As we can see, outputs of the **bash script file** and **cat list.txt** command are the same.

- There are two ways to write in a file. First one is to use a single greater than (>) symbol. When we do this, any input we provided will be written in that file. If the file already exists, it's going to be overwritten. Nevertheless, we can append things into that file by using double greater than (>>) symbols. Let's see an example.

```
#!/bin/bash

echo "End of the list" >> list.txt

sys_info="Current Directory: $(pwd) Current User: $(whoami) Current Date & Time: $(date)"

echo $sys_info > sysinfo.txt
```

→The End of the list sentence has been **appended** into the list.txt file whereas output of pwd, whoami, and date commands has been added into a non-existed file.

Task Automation

Scheduling Jobs

-In Linux, we can leverage scheduling and automating tasks by using bash. We can either create bash script files or use specific commands for scheduling tasks. For instance, we might need to automate tasks to be performed when we are away from keyboard. Beside of creating bash script files, we can leverage **at** and **cron** services.

- **at:** This command is used to schedule a command to run *once at a particular time* (*one-time task*). Such as updating the system **at 5 P.M**. The at command has several options. For instance, the **-m** option sends mail to the user, **-M** option prevents sending mail to the user, **-f** option read a job from specified file, **-t** option runs the job at the specified time value, **-v** option displays the time of the jobs that will be executed. Additionally, we can use a couple of methods when providing time for the tasks. For example, we can create a task to be performed at *12 P.M. today* by using **at 12PM** or **at 12.00** commands. To view the scheduled tasks in queue, we can use **atq** command and to delete a scheduled task, we can use **atrm** command.
- **atq:** This command is used to *view the current queue of tasks* scheduled by *at command*.
- **atrm:** This command is used to *delete a scheduled task* scheduled by *at command*.
- **Cron Daemon:** This daemon is used to manage scheduled tasks called *cron jobs*. These cron jobs will happen every single time at *repeating interval*. The cron daemon checks its crontab configuration file every single time to discover whether there's any tasks that need to be accomplished; and if there is any, it's going to be executed. The cron daemon is controlled by using the **crontab** command.
- **crontab:** This command is used to create, view, and delete *crontab files*. It has several options. Such as, the **-e** option edits the crontab for current user, **-l** option views crontab for current user, **-r** option deletes current crontab file, **-u** option creates a crontab file for specified user.

-When we run **crontab -e** command, it'll open up a text editor to define cron jobs. Let's see how we can define cron jobs.

- First of all, the format of cron jobs is as follows:

  *** * * * * /path/to/command**. The five asterisks (*) represent the schedule in this order from left-hand side to right-hand side:

  -Minute (0 - 59)

  -Hour (0 - 23)

  -Day of month (1 - 31)

  -Month (1 - 12)

  -Day of week (0 - 6, Sunday = 0 or 7)

- Let's see some examples.

  *** 22 * * 1-3 /home/kali/Tools/scan_system.sh** → This cron job will run the *scan_system.sh* file at *10 P.M. from Monday through Wednesday*.

  30 12 15 * * /home/kali/Tools/check_storage_units.sh → This cron job will run the *check_storage_units.sh* file at *12.30 P.M. on 15th day of every month*.

-The scheduled task information can be found in **/etc/cron.d/** and **/var/spool/cron/** directories. The *root user* and *services* can use the */etc/cron.d/* directory to schedule system-wide tasks. However, the *regular users* are *not allowed* to populate the /etc/cron.d/ directory, they can schedule tasks in personal directory located underneath */var/spool/cron/* directory. Additionally, there are directories like **/etc/cron.yearly/**, **/etc/cron.monthly/**, **/etc/cron.weekly/**,**/etc/cron.daily/**, **/etc/cron.hourly/**.

Hands-on Experience – at, atq, atrm, crontab

-We can create one-time task by using **at** command.

```
┌──(root㉿kali-purple)-[/home/whiterose/Desktop]
└─# at now + 2 hours
warning: commands will be executed using /bin/sh
at Fri Oct  4 16:18:00 2024
at> rm /home/whiterose/Desktop/bash.sh
```

- In this case, the */home/whiterose/Desktop/bash.sh* file will be *removed in 2 hours*.

-We can list the queue tasks by using **atq** command.

```
┌──(root㉿kali-purple)-[/home/whiterose/Desktop]
└─# atq
4       Fri Oct  4 12:00:00 2024 a root
3       Fri Oct  4 12:00:00 2024 a root
2       Fri Oct  4 23:40:00 2024 a root
1       Sat Sep 28 16:00:00 2024 a root
```

- In this case, there are *four tasks* in the queue.

-We can remove task from the queue by using **atrm** command.

```
┌──(root㉿kali-purple)-[/home/whiterose/Desktop]
└─# atrm 2 ; atq
4       Fri Oct  4 12:00:00 2024 a root
3       Fri Oct  4 12:00:00 2024 a root
```

- In this case, the *task labeled as 2* has been removed from the queue.

-We can create a task to be performed in repeating interval by using **crontab** command and **-e** option to edit crontab file.

```
30 12 * * 1 /bin/rm /home/whiterose/Desktop/remove_me.txt
```

- After running the crontab -e command, crontab file will open with a text editor (vim or nano). In this case, **12.30 every Monday**, **/home/whiterose/Desktop/remove_me.txt** file will be removed. Also, notice that we need to reference the path of command we'd like to use.

Implementing Version Control Using Git

-When developers *collaboratively* build a project, they need to have a **distributed version control system**. Using a distributed version control system has some benefits, such as *minimizing conflicts*, *managing different versions of the project easily*, *reverting back to the older version* if there are bugs etc. In this case, we can use something like **Git**.

- **Git:** Git is a mature and actively maintained *open-source project*. It was developed in 2005 by *Linus Torvalds* whom the person behind the development of Linux operating system.
- **Git Repository:** Git repository is a storage area where the versions of code and related files are stored. Git repository is the core component of Git.

-We can use package managers to install git in our system. For instance, on Debian-based systems, we can use **apt** (**sudo apt install git**).

- **git:** This command provides us to *interact with Git and Git repositories* through command line interface. It has several subcommands. Such as, **config** subcommand sets options for repository or Git users, **init** subcommand creates Git repository or reinitialize an existing one, **clone** subcommand creates a copy of an existing repository, **add** subcommand adds files to be tracked by Git repository, **commit** subcommand updates the Git repository with changes, **status** subcommand displays status of the repository, **branch** subcommand manages branches (after changes), **merge** subcommand integrates changes into a master branch, **pull** subcommand acquire and merge changes, **push** subcommand uploads local working copy

of a repository to a remote repository, **log** subcommand displays the changes made (local repository), **checkout** subcommand switches to a specific branch, **tag** subcommand adds a tag to our git repository.

-Suppose that we need to create a local repository and commit some changes to it. Let's look at the process of doing it.

1. Configure global settings including username and mail address by using **git config --global user.name 'username'** and **git config --global user.mail 'mail-address'** commands.
2. Create a directory where the project will reside by using **mkdir /project** command.
3. Change into the created directory with cd command and initialize it with Git to designate it as Git repository. To do this, use **git init /project** command.
4. Add project files to the repository. To do this, use **git clone /project** and **git add project-file** commands.
5. Commit the changes by taking a snapshot of the project by using **git commit -m 'initial commit'** command.

-We can view the status of our git repository by using **git status** command.

- **Branching:** Git branch is a feature that is available in most modern version control systems. Branchers are like a pointer to a snapshot of the different changes in our project. By creating a branch in the master copy of the code, we are able to work on that *without affecting other developers*. To do this, we need to use **git branch new-branch-name** command. After we're done, we need to merge the changes into the master branch by using **git merge new-branch-name** command. If we want to delete the branch after merging, we need to use **git branch -d new-branch-name** command.

-The *collaborative feature* of Git is game changing. If we're collaborating with other developers, we can *pull their changes* and *push our own*. To pull changes from a branch, we need to use **git pull other branch** command. To push our changes from local to the central repository, we need to use **git push <remote-repository> mybranch** command.

-We can view the commit history of our repository using **git log** command. For instance, **git log --since=10.days** command will display the commit changes within the last ten days.

-We can navigate and switch between branches of a project using **git checkout specific-branch-name** command. This allows developers to focus their attention on different branches of the same project.

- **.gitignore file:** This file identifies files that should be ignored during a commit action (temporary files etc.). It exists within the repository.
- ***.git files:** The files with .git extension within the repository contains all files that Git uses to manage version control for a project.

Orchestration

-Orchestration processes can be used to work more effectively rather than ordinary automations (at, cron, bash). Imagine that as a system administrator, we need to deploy, manage and maintain hundreds, maybe thousands of servers and it can't be done by only using at, cron or bash scripts.

- **Orchestration:** Orchestration is known as the *automated configuration*, *management* and *coordination of computer systems*, *applications* and *services*. Orchestration provides system administrators to manage complex tasks more easily.
- **Automation:** Automation is the *process of accomplishing a configuration task* without human intervention.

-Automation refers to a *single task* whereas orchestration refers to managing a *larger scale of series of tasks*. Basically, orchestration is the *automation of the automations*.

-One of the benefits of cloud computing is *rapid elasticity*. Rapid elasticity computing would not be possible without orchestration.

-There are three main types of orchestration. Resource orchestration, Workload orchestration, Service orchestration.

-When using orchestration for our complex tasks, we don't have to stick to one vendor. We can use multiple vendors considering our needs. There are third-party orchestration platforms that protect us from vendor lock-in. Also, some tools are designated to provide orchestration. Let's take a look at couple of them.

- **Chef:** This tool uses 'cookbooks' to deliver configuration declarations to cloud and on-premise managed systems.
- **Puppet:** Puppet uses manifest files to define infrastructure as code (IaC) for application, cloud, and infrastructure orchestration. It's similar to Chef.
- **Ansible:** This tool uses YAML files to create repeatable 'playbooks'. Unlike Chef and Puppet, Ansible doesn't use user agents. It's developed by RedHat.
- **Docker:** Docker is an open platform for developing, shipping, running, and deploying applications using container-based virtualization.
- **Kubernetes:** Kubernetes provides container deployment and application orchestration for cloud and on-premise container environments.
- **OpenStack:** OpenStack was originally deployed as an IaaS (Infrastructure as a Service) solution to manage cloud resources.
- **GitHub:** GitHub is a service that allows developers to share their code.

-Let's explain agent-based and agentless orchestrations in order to find out differences between them.

- **Agent-Based Orchestrations:** Agent-based orchestration tools *require a software component* to reside on the managed device.
- **Agentless Orchestrations:** Agentless orchestration tools *do not require additional software* to exist on the managed system.

-When we are doing orchestration, we need to be familiar with couple of concepts besides the tools.

- **Orchestration Attributes:** Orchestration Attributes is used to *identify specific configurations* that need to be set through the orchestration process.
- **Inventory Management:** Inventory management of hardware, virtual machines, operating systems, applications, configurations can all be managed through orchestration tools. It's crucial for effective administration.

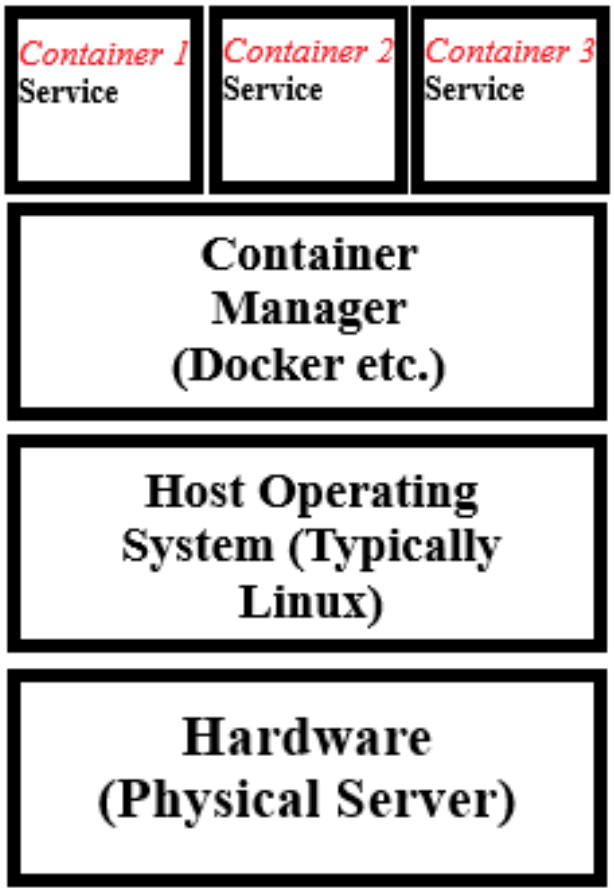
- **Configuration Management:** Configuration management ensures that consistently configured system, enforced security, service-level agreements, and efficient change management.

Containerization

-In today's world, the popular type of containerization is known as *container-based virtualization* and also referred to as *containerization*. Containerization is more focused on *servers* than user workstations. With this type of virtualization, the operating system kernel space is going to be shared across multiple virtual machines, however the user space is not.

- **Containerization:** Containerization is a type of virtualization applied by a host operating system to provision an isolated execution environment for an application. Containerization is considered *fairly secure* because it enforces *resource separation* at the operating system level. Containerization is commonly used with Linux servers. Examples of containerization software include *Kubernetes*, *Docker*, *Parallels Virtuozzo*, *OpenVZ*.

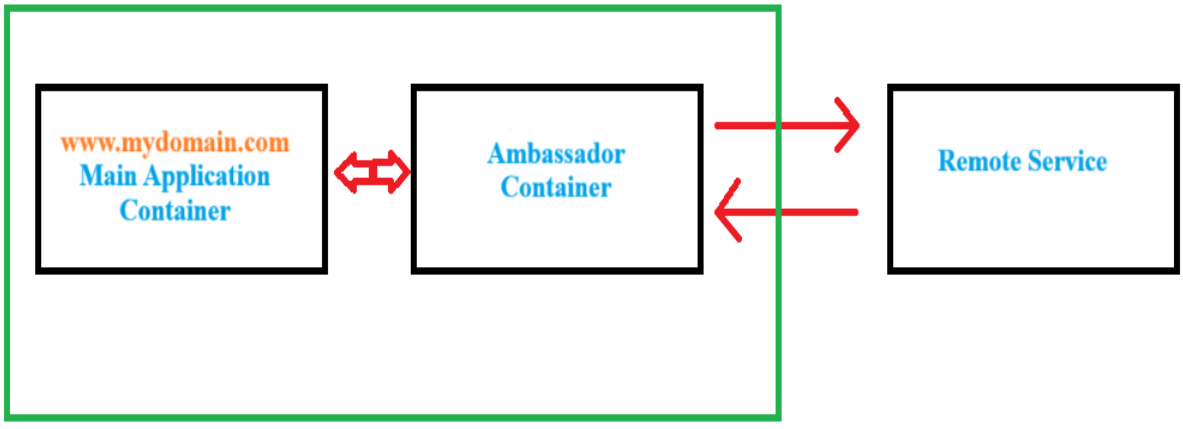
-Let's explain how containerization looks like. There is a hardware, and we got **host operating system** on top of that. Then, we have **container manager** like Docker and this container manager is used to create different containers that have *different applications/services* within them. On the image below, we can observe the structure visually.



-Rather than installing operating system using virtual machines, containerization takes *a lot less resources* (hardware). In case of using virtual machines, it would add *ten to fifteen gigabytes extra space* since each container would have own operating system. Additionally, since containers has been *logically isolated*, they **cannot interface with each other**. If we want to establish communication between two containers, we need to connect them *through a virtual network*. The isolated environment of containerization enhances *security*. However, if a threat actor *compromises the Host operating system*, they are able to *access all of the containers* inside of the host operating system.

-When we deploy Kubernetes, we will gain access to a cluster that contains a set of worker machines which is called **nodes**.

- **Node:** Node is an object that runs the containerized applications. Inside of these nodes, we host something called **pod**.
  - **Pod:** One or more containers that have *shared storage* and *network resources*. Pod is the name used by Kubernetes. In addition to pod, there is something called **sidecar**.
  - **Sidecar:** Designed to run alongside the main container or pod. Benefit of using sidecar is that we can *enhance* and *extend the functionalities of the main container without modifying its codebase*. Also, sidecar can also use different programming languages than the main container does and it's a good feature if we want to build **microservices**. In addition to sidecar, there is also a specialized type of sidecar known as an **ambassador container**.
  - **Ambassador Container:** A special type of sidecar container that simplifies the process of accessing data and services outside of a given pod. It's also used to hide the complexity involved in accessing external services by providing us with a uniform interface to access those services. We can see the role of ambassador container visually below.



-The ambassador container is being treated as proxy between our main application container and the remote service.

-Another area that we can take advantage of containerization is storage. One thing we need to worry about is that containers are designed to be *non-persistent* by default, so we need to create a *persistent storage volume* to eliminate this issue. We want to create this volume on our physical storage device, named as **PersistentVolume**, then we want to create a **PersistentVolumeClaim** that'll automatically be bound to that suitable PersistentVolume. Whenever we create a pod, we can configure it using PersistentVolumeClaim and access this persistent storage for any data that needs to be read or written to a more permanent storage media area outside of a given container.

-Keep in mind that Docker and Kubernetes can work together without any problem even though they may seem like competitors. We can use something called **docker compose** to accomplish these kinds of situations. Thanks to the *docker compose*, we can do all the necessary configuration for running a *multi-container application within a single file*. Then we can convert it using to Kubernetes by using something called **Kompose**.

- **Container Registry:** Container registry is a place to *store*, *manage*, and *secure container images*. From within the registry, we can conduct *vulnerability analysis* of our containers and implement *fine grain access controls* to those containers. Container registries are set up to work with *Docker containers*, and these registries can be linked from within Kubernetes to allow us to pull an image from the registry and launch the container using automations and orchestrations.
- **Kubernetes Service Mesh:** Kubernetes service mesh *manages the network traffic* between different services, containers, and pods. Generally, the service mesh is going to be implemented as a *set of network proxies* that are deployed alongside a sidecar, and they can make use of the *ambassador containers*. Service mesh sits on top of a Kubernetes infrastructure to provide *interservice communication over the network*.

Container Networking

-Container networking represents the capability of communication status of containers. Containers might be able to communicate with other containers, some of them able to communicate with even host. Let's examine the network connection types used in containerization platforms like Docker and Kubernetes.

- **None:** This type of network connection is the simplest. Because the container *only has a loopback interface*. Also, there is **no connectivity** with an *external network.*
- **Bridge:** It's an internal host network connection type that provides communication *between containers on the same host*. Containers *cannot be accessed from outside the host*. Bridge network is the **default for Docker** containers.
- **Host:** In this type of network connection, the container *shares its network namespace with the host* and it enables *high-speed connection*.
- **Underlay:** The underlay network connection exposes *host interface to containers running on the host*. It also removes the need for port-mapping which makes it *more efficient than bridge network connection*.
- **Overlay:** It ensures that containers connected to *two different overlay networks isolated from each other*. It uses networking tunnels to communicate across host. In overlay network connection, two different overlays cannot communicate with each other over the *local bridge*.

Container Operations

-We can use a program like Docker or Podman to run and manage container images. Podman is a containerization program that runs on RHEL (RedHat Enterprise Linux) and supports almost the same features in Docker without using the Docker itself. Let's examine the commands for *configuring container images*.

- **podman build:** This command allows us to build a container image by using a *Docker file instruction* located in the current directory.
- **podman push:** This command allows us to push a container image to a specified destination. This destination could be any container management service like Docker and Kubernetes.

- **podman pull:** This command allows us to pull a container image form a container registry. This registry could be public or private registry.
- **podman images:** This command lists out the container images available on the local system.
- **podman ps -a:** This command lists out all running container on the screen.
- **podman rmi <image-id>:** This command allows us to remove the specified container image.

-Keep in mind that, **syntax of Docker and Podman are the same.** Let's examine the commands for *managing containers*.

- **podman start <container image name>:** This command allows us to start the specified container.
- **podman stop <container image name>:** This command allows us to stop the specified container.
- **podman inspect <container image name>:** This command allows us to inspect the specified container. It will display the configuration of a given container/image.
- **podman pod list:** This command list all the containers on the system.

-Let's talk about how we can *deploy images*. Best way to deploy an image is to use a template. A template file is going to be created based on the format of the containerization solution that we're going to be using like Kubernetes. This file will include a unique user-friendly name as well as descriptions of the containers and the pods that are being used by that image.

-Now, let's take a look at how we can connect containers.

- **podman attach <container image name>:** This command allows us to connect to specified container in order to interact with them .

-Let's see how to perform log actions on containers.

- **podman log <container name>:** This command allows us to gather the logs from a specified container.

-Let's see how to map ports on the containers.

- **podman port <container name>:** This command lists all of the port mappings for specified container.

- **podman create --expose=<port-number>:** This command allows us to create a new container with a specified port exposed on the host system.
- **podman run --expose=<port-number>:** This command exposes the specified port on a container when we start it up.

Sandboxed Applications

-Sandboxed approach to software development and management creates a *safe environment* for running and testing the project that keeps users and environment safe. Sandboxing *isolates the environment* and *protect system resources* from malware and other cyber threats.

-Application sandboxing helps us to *improve security* by isolating and shielding the application from outside intruders and malwares. Sandboxing is useful approach when running applications/files from *unknown* or *untrustworthy* resources. Sandboxing also *increases application integrity* because it lets developers wrap the application in security policies and isolate and protect the application within its own environment.

-To provide sandboxing in Linux, we can use snapd (snap daemon), Flatpak, or AppImage.

- **Snap:** The snap is a bundle that contains an app and its dependencies that work without modification across all Linux distributions. To manage and maintain Snaps or applications, we need to run the snap or snapd command.
  - ***snap:*** This command is used to find a snap or application to install. For instance, if we enter **snap find "media player"** command, we will get a list of media players available to download as snap bundle. We can install applications by using **snap install <application-name>** command. By default, applications are installed under the **/snap/bin/** directory. We can update applications by using **snap refresh <application-name>** command.
  - ***snapd:*** The backend daemon that runs the snaps on a system. Keep in mind that snapd *doesn't come pre-installed* by default, so we have to install it using package managers like yum. After installing, we need to enable network communication for the snapd utility by using **systemctl enable --now snapd.socket** command.

- **Flatpak:** Flatpak runs in a sandbox (isolated environment) that contains everything needed for the programs to operate. It lets us distribute, install and manage software without need to worry about the dependencies, runtime, or specific Linux distribution that's being used.
- **AppImage:** AppImage is another application containerization/sandboxing tool. It's a universal package manager where the apps are installed without modifying system libraries or system preferences. With AppImage, we simply download the packages that we want to run and then we run them without installing them. This is known as **Portable Apps**. Since they are portable, we can run them from anywhere of our operating system. It's recommended to create a directory under the /home/ directory to put al the AppImage applications.

Infrastructure as Code (IaC)

- **Infrastructure as Code (IaC):** IaC is a provisioning architecture in which deployment of resources is performed by *scripted automation* and *orchestration*. IaC is *secure* because it allows for the use of scripted approaches to provisioning infrastructure in the cloud. IaC comes down to three key areas. *Templates*, *scripts*, *policies*.

-*Robust orchestration* can lower overall IT costs, speed up deployments, and increase security.

-One important thing when applying IaC is that because we're using templates and standardization for everything, there are people who want to have their *special snowflakes*.

- **Snowflake System:** Any system that is different in its configuration compared to a standard template within an IaC architecture. Issue is, it causes security, configuration, and configuration problems.
- **Idempotence:** A property of IaC where an automation or orchestration action always produces the same result, regardless of the component's previous state.

-IaC uses carefully developed and tested scripts and orchestration runbooks to generate consistent builds. For that reason, we shouldn't allow special snowflakes unless there is a really solid reason.

-When conducting IaC, we use some specialized software. Let's see what we can use.

- **Terraform:** Terraform software is a modern method used to provision, change, and version resources on any cloud-based environment using automation and orchestration. It's commonly used in IaC and multi-cloud deployments, Kubernetes management, Network infrastructure, virtual machine images, policy as code enforcement.
- **SaltStack (Salt):** SaltStack is a configuration management and orchestration tool which is commonly used with IaC deployments. SaltStack eliminates the manual processes used by legal IT operations.

Continuous Integration & Continuous Deployment (CI/CD)

-CI/CD concepts enable us to speed up the workflows.

- **Continuous Integration (CI):** Continuous integration is a software development method where code updates are tested and committed to development or build server/code repository rapidly. Continuous integration can test and commit updates multiple times per day. Continuous integration detects and resolves development conflicts early and often.

-There is also a term called Continuous Delivery. Let's explain it.

- **Continuous Delivery:** Continuous delivery is a software development method where application and platform requirements are frequently tested and validated for immediate availability.
- **Continuous Deployment:** Continuous deployment is a software development method where application and platform updates are committed to production rapidly.

-In summary, continuous delivery is focused on *automated testing of code* in order to get it ready for release. Continuous deployment is focused on *automated testing* and *release of code* in order to get it into the production environment more quickly

References

- https://www.udemy.com/course/comptia-linux/
- https://www.geeksforgeeks.org/types-of-user-profile-management-in-linux/
- https://www.redhat.com/sysadmin/linux-file-permissions-explained
- https://www.redhat.com/sysadmin/suid-sgid-sticky-bit
- https://www.geeksforgeeks.org/disk-partitioning-in-linux/
- https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/9/html/configuring_and_managing_logical_volumes/overview-of-logical-volume-management_configuring-and-managing-logical-volumes
- https://ubuntu.com/server/docs/about-logical-volume-management-lvm
- https://www.redhat.com/en/topics/linux/what-is-the-linux-kernel
- https://www.redhat.com/en/blog/architecting-containers-part-1-why-understanding-user-space-vs-kernel-space-matters
- https://linux-kernel-labs.github.io/refs/heads/master/labs/kernel_modules.html
- https://www.techtarget.com/searchnetworking/definition/Preboot-Execution-Environment
- https://www.geeksforgeeks.org/public-cloud-vs-private-cloud-vs-hybrid-cloud/
- https://www.redhat.com/en/topics/virtualization/what-is-KVM
- https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/8/html/packaging_and_distributing_software/introduction-to-rpm_packaging-and-distributing-software
- https://www.redhat.com/sysadmin/how-manage-packages
- https://ubuntu.com/server/docs/package-management
- https://nordvpn.com/blog/what-is-openvpn/
- https://www.redhat.com/en/topics/linux/what-is-selinux
- https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/4/html/reference_guide/s2-selinux-files-etc
- https://www.angelfire.com/mi/genastorhotz/reality/computers/linux/bashmetachars.html
- https://www.atlassian.com/git/tutorials/what-is-git

- https://www.atlassian.com/git/glossary#commands
- https://www.redhat.com/en/topics/automation/what-is-infrastructure-as-code-iac
- https://www.vmware.com/topics/container-networking
- https://www.tigera.io/learn/guides/kubernetes-networking/container-networking/
- https://learncloudnative.com/blog/2020-10-03-ambassador-pattern
- https://en.wikipedia.org/wiki/Penguin#/media/File:Aptenodytes_forsteri_-Snow_Hill_Island,_Antarctica_-adults_and_juvenile-8.jpg