-- Day3 --

***Everything in a Linux is a FILE!

There are **3** roots I the **Linux Operating System**:

1: **`root`** user, **`uid`** = **0**

2: **`/`** directory, root file system

3: **`/root/`** directory, root home directory

`cd` command can work with the:-

1: absolute/full path

ex: **`cd /home/work/filenme`**

2: relative path:

ex: **`cd .`** : current working directory

ex: **`cd ..`** : parent working directory

*** **`cd ../..`** to get the root, will work as **`cd ../../../../../../../`** .. etc

because the root directory doesn't have a parent, it is the parent of itself

---

-- Day 4 --

General Formula of any Linux Command:

`command [options] [arguments]`

- an option changes the behavior of the command.
- an argument gives additional info to the command

*** Don't use **`rm -rf`** command, use it very carefully!!

*** Every file in Linux started with **`.`** is a *hidden file*, like **`.file1`**

*** **`exit`** = **`logout`** = **`ctrl + D`**

*** **`reset`** = **`ctrl + L`**

*** using **`bg`** to continue the paused commands and processes, make the **`bg`** command not able to be paused again

*** **`shutdown -r now`** = **`systemctl reboot`** = **`reboot`**

*** **Users & Groups** ***

When a user created, automatically a "primary group" with its name is created and it will be its private group!

A user can be member in more than a group, but only secondary groups!

A user can be in one primary group only!

U can't use the user till you set him a password!

A normal user can't set a weak password for himself, but only the root can!!

-- Day 5 --

A normal user has no access to the system, except his home directory

The **root** user has access to the whole System with no limits!

***

When adding a user to another secondary group (while he is in another secondary group):

- We use the command: **`usermod -a -G [groupname] [username]`**
- without adding **`-a`** option to the command, the user will be deleted from any other secondary group,
  (except his own secondary group) and added to the new selected.
- To delete all the secondary groups for the user, we use the command (**`usermod -G username username`**)
- While the username is the same, it will delete all other secondary groups except its own private group with the same name!

***

There is a big difference between the **`userid`** and **`groupid`**:

Primary group is so important for Ownership!

Because when you create a file, the file' ownership goes to the user and its primary group

→ When changing the primary group of an existed user, it is deleted from the ex-primary group,
and also deleted from it as a secondary group, **IF and only IF**, the changed group was not the same private group of the user!!

→ It doesn't make any sense to make the user private group has more than one user!! It is PRIVATE!

→ Now we **cat** the **`/etc/passwd`**: → Where all Users IDs and information stored on the System

`user:x:uid:gid: :/home/user:/bin/bash`

- **`x`** here means the password, but it is hidden and hashed in another file, **`/etc/shadow`**
- The space between the home directory of the user and its primary group, is for the user description, It also called "**GECOS**"
- The **`/bin/bash`** in the end, is the user shell!

Passwords are hashed in **MD5** due to RHEL 5 and CentOS 5, now we use **SHA**.

There are many shells/terminals in Linux, but the default one is **`/bin/bash`**

→ Now we **cat** the **`/etc/gshadow`**: → Where all groups passwords are stored.

`groupname:!: :members` → from: # **`cat /etc/gshadow`**

The exclamation sign **!** means there is no password set yet

The space between the group members and the hashed password, means the group admin!

***

When you delete a user, his files still existed in his home directory!

After that you can delete his home directory, **`rm -rf /home/userdel/`**

Or you can only use: **`userdel -r [username]`**, just to delete the user and his home directory.

***

Note: If the user1' primary group is also a primary group of another user2, and you delete the user1,

The primary group won't be deleted, as it is a primary group of another user!

This would happen unless it was a secondary group!

***

When you are asked to add a user to group1, group2 and group3:

If you're not told to add it to a primary group, then all groups are secondary groups!

***

If you add ex-user and his primary group was still existed, the add process will fail!

Because you must choose his primary group first using: **`usermod -g [primarygroup] [username]`**

***

A new user created can't login in the terminal until you set it a password!

The user will still be active on the system, and you can switch to it from the root user.

***

NOTE: If we have deleted a user, and we list his own files, we will see that the Ownership of these files will move to a numeric value of a user id which is not still existed.

If we then added a new user to the system with that numeric userID, the new user will take the Ownership of these files!

→ You can delete all the files started with **.**, the hidden files, with one command (**`rm -f .*`**).

Permissions:-

/* There are three things you can do to files in Linux:

You can **r** = **read**, **w** = **write**, **x** = **execute** */

There are 3 persons who can deal with files on a system:

1) Owner/User (**r, w, x**)

2) User's group (**r, w, x**)

3) Other/Default Category (**r, w, x**)

Any one of U,G,Other can (read, write, or execute)

This is not the *Actual Permissions, but the *Probability of permissions due to the Policies.

*****

Every permission can be represented by **1** bit, so we get **3** bits to all of them (User, Group, Others)

- rwx rwx rwx

The **10**th bit is represented with the file type, and is the first bit of the **10** bits!

- = normal file

d = directory

b = block device

c = character device

l = link file

*****

→ Now if we **ls -l**, to any directory or file :-

The result we get: **-rw-rw-r--. 1 root root 1562 Jan 22 2024 work**

- If the permission is not given in any bit, we represent it by **-** sign!
- The **.** sign in the end of 10 bits, means that **SELinux** is protecting the file!
- The number after the permissions and . sign, means the number of links/shortcuts of the file (link count)
- After the number came the user, group, file size, last accessed time and file name
- If we list any other type except normal files, the color of the filename will be different!
- Link file means shortcut on windows!

There are two types of device files:

1) Block device: the device which can be read/write to, block by block or chunk by chunk! like the hard-disk!

2) Character device: the device which can be read/write to, character by character! like the keyboard or the tty!

tty is represented on Linux as a character device file on the **/dev/tty**

Color of the files in terminal:

1) White: Normal file

2) Blue: Directory

3) Lemon: Executable file

*****

Executable file means that you can put code, binary or shell in the file and execute it!

It looks like the **.bat** or **.exe** in windows!

*****

You can make a combination with **chmod** command!

Ex: **chmod u=rwx,g=rx,o=r** OR **chmod u+x,g-x,o-wx**

If you don't select the permission to whom (u,g,o), it will execute it to all of them, Ex: **chmod rwx file**.

*****

When you apply a permission on a directory, It won't be applied to the files inside it!!!!

Or You can use the **-R** option to the **chmod** command, **chmod -R**

*****

In copy command "**cp**", we use **-r**, but in "**chmod**" command we use **-R**, both for recursively!!

*****

Meaning of Permissions:

File **–** :

- **r** → to read or view its content
- **w** → write/edit/delete/overwrite
- **x** → execute or run the file

Directory **d** :

- **r** → to view the content inside the directory, to make (**ls**)
- **w** → to add/delete files, or delete the directory, like using (**rm**, **cp**, etc..)
- **x** → to make (**cd**) or (**ls -l**) to the directory

-- Day 6 --

Permissions:-

- We can append two permissions in one command like:

ex: **chmod u+r,g-w,o-wx file1**

→ There are **3** ways of Editing **Permissions**:-

- The Method of writing permissions is named: /**Symbolic Method**/ = Using symbols!
- We can use **=** to set permission, It is called **Setting Permissions**.

Which means, forget everything and set these permissions as new permissions

ex: **chmod u=rwx,g=rx,o=rx file2**

- Another Method called /**Numeric Method**/ = Using numbers only!

Here we represent every permission with a number:

r = 4

w = 2

x = 1

The addition of any combination of these numbers will not ever conflict with other!!

Remember: **chmod [ugo] [rwx] [filename]** ?

In the /Numeric Method/: we get the addition of the permissions we want to give to the UGO, and then write it in its bit,

like: user=rwx=4+2+1=7 , group=rw=4+2=6, o=rw=4+1=5

So we here get user permissions = 7, group = 6, other = 5

Note: There is no any overlap with any permission and the other!!

ex: **chmod u+rwx,g+rw,o+rx textfile → chmod 765 textfile**

Also Remember: We can use **-R** to recursive the command to the whole directory with its content!!

End of Permissions :) !

****

\\ Redirection //

In Redirection we use three things: Input, Output and Error

- Your default Input is **Keyboard**, and the default Output is the **Screen**!

- each one is represented by a number:

Input = **0**, Output = **1**, Error = **2**;

Each number is called a *descriptor*!

We use also **2** symbols: to represent whether it is an input, output or error!

(**<**) to input, (**>**) to output/error

NOTE: Error is an /Exceptional case/, so it must be passed

Only Input or Output which could be passed by default!!

- Sometimes one command can result both Errors and Outputs!!

"**cat**" command takes its input from a file by default, and doesn't take it from the Keyboard!!!

- **cat file.txt**, do the same as **cat <file.txt**

- To send input to the '**cat**' command, use: **cat <<EndOfText>> filetest.txt**, to append the input in the file

- '**EndOfText**' here means, add the inputs to the file and when you reach this word, terminate the input operation!!

- We can take input from a file to another file, ex: **# cat <file1> file2.txt**

- To append the input of the file we use the **cat** command like: **# cat <file1>> file2.txt**

WARNING: DO NOT try to make like this: (**cat <file >>file**), this command will work infinite, as It takes the input from the file and put it in the same file, and it will still look to its content as an input because it will still adding the same data to it.

******

- The name of the file doesn't have any value in Linux!

- What's matter only the content inside the file!!

ex: # **cp file2 music.mp3**

file music.mp3

music.mp3: ASCII text

- Every file has a signature, it looks like a header to the file which tells what its type!!

- Linux only looks for the file' content not the file name or its extension!

→ To put the error and the output in the same file, There are **2** ways:

1) **`ls file1 file1001 >>outerror.txt 2>>outerror.txt`**

2) use '**`&>`**' to append both output and error

- The difference between '**`less`**' and '**`more`**', is that **`more`** terminates after reach the end of the file, but **`less`** needs to '**`q`**' quit

→ We can redirect the output as an input to another command using '**`|`**' pipe command, ex: **`ls -lR /usr/ | less`**

- '**`tee`**' always overwrite the file of the results,
- but we can use the '**`-a`**' option to append the result of the command to the file

*****

Remember: TTY is the terminal when you have physical access to the machine

→ **PTS** = Psudo terminal screen = Terminal Emulator

as you emulate that you have a physical access or console access on the machine, but you connect remotely!!

Connecting over **telnet**, **ssh**, or the **terminal** opened in the graphical user interface.

`:0` refers to the first graphical monitor!

`:1` refers to the second, and **`:2`** the third ... etc

→ **GDM** = Gnome Display Manager: The Graphical user interface of the user in Linux.

GDM is the only service-user is logged in!!

→ **`whereis`** command gives the result of: Binary of command, Its man page, and the documentation.

-- Day 7 --

*** **Kernel** is dealing with everything in Linux *with numbers* ***

Raw Space or Raw Disk: Is the new disk you recently bought

MBR : **Master Boot Record**

MBR is necessarily to be existed, to manage and control the Disk Space

It is used to know and map where the partition Begin and End.

MBR, is not the only way to manage and control the space on the disk, there is also GPT.

MBR: is consisted of 3 things:

1) **Partition Table**: to know the Begin and the End of partitions!

2) **Boot Loader**:

3) **Magic Number**:

* If the Partition table or MBR is corrupted, the Disk will get back 'Raw' and the data will be lost!!

→ **FileSystem:**

- A ***filesystem*** is the methods and data structures that an operating system uses to keep track of files on a disk or partition; that is, the way the files are organized on the disk.
- Is the way of partitioning every partition.
- FileSystem organize and manage the Store and Retrieve of data on the Partition.
- You can choose the way of partitioning due to your needs!
- Every filesystem has both advantages and disadvantages

→ **FAT32:**

- The file size can't be bigger than **4**GBs
- It doesn't support Encryption or Compression
- Supported on any Operating System

→ Other examples:
NTFS, ext2, ext3, ext4, XFS, ZFS, BTRFS

- Every Partition is a block of blocks
- Every Partition has an 'inode table'

→ ***inode table***: is a table that tells all information about the file, where the block it stored and its privileges etc.

- *inode number* also called *index number*.
- You can call it the 'Metadata' about the files, Or 'Data about Data'
- Metadata is stored in the inode table

NOTE: Partition Table stores the partition info, not the Metadata itself!

- Partition Table is for the whole Disk, but the inode table is only for the filesystem/partition

- inode table also existed in **Windows**, but with different structure.
- If inode table is corrupted, it affects only its Partition.
- If Partition table is corrupted, the whole Disk will be affected.

→ **Format**: is the process of creating the Filesystem for every partition.

Directory: is a 'special' file, pointing to other files.

→ These are all the metadata stored about a File in the inode table:

- inode number (the number of the block where the file stored)
- permission
- owner
- access time
- creation time
- modification time
- link count

→ These are all the metadata stored about a Directory in the inode table:
you'll see the same info mentioned above,
but with a metadata (pointers) about the files with their numbers which the directory is pointing to.

→ Every inode number is pointing to a block:
- By default, every inode number is pointing to a block with **128**-byte size.
- We can change the size of blocks while Formatting the Disk demanding on the Filesystem and what will be stored on it.
- Every file is pointed to by only **1** inode number.

→ If the inode number is pointing to **50**-byte file size, and you're storing files with **128**-byte file size:
the rest of the space will be wasted!
- If the files size is larger than the block size, The file size will be separated into a suitable number of blocks, until it is enough to the file size, and the inode number will point to the **1st** first block of these blocks and the rest of blocks will be flagged as it is '**used**'.

→ **De-fragmentation**: is the process of assembling the Miscellaneous Units/Blocks to the same place to reduce the seek time to make the reading and writing of files faster.

- When you do the disk free space (**`df`** command), you'll find real and virtual filesystems.

→ It is very important to know about the Inode numbers,
- Sometimes when you try to copy or write files on a free space on disk, you are surprised that the process fails!
- That's because the Free space already existed, but there aren't enough Inode numbers to point to it!!
- This also mean that you made the block size on the filesystem big, or every Inode number is pointing to a very large space!!
and you stored a small-size files, so now you have a very big wasted space..
- In this case, you must be able to know the Inodes, and see how many used and available
- To ensure the issue is from the Inode or it is not!

-- Day 8 --

Continue Inodes:

→ It is impossible that **2** different blocks have the same Inode number,
because this represents a physical place on the disk

→ When you copy a file on the same Filesystem (Partition), It reserves an Inode number and then takes a copy of the data blocks and write it on another data blocks, and makes a Pointer to point to it in the directory
- When you cut/move a file on the same Filesystem (Partition), already written data blocks still as it, and the Inode number also!
and what changes is the *Reference* or the *Pointer* which is what was pointing to the data blocks in the directory!!
- Briefly, the data blocks still as it and also the Inode number, but just changing the *Pointer* from a directory to another.
- That's why the Move is so fast!

→ When you copy a file on a different Filesystem (Partition), It reserves an Inode number and then takes a copy of the data blocks and write it on another data blocks on the other Partition, and makes a Pointer to point to it in the directory.

- When you cut/move a file on a different Filesystem (Partition), It reserves a new Inode number and then takes a copy of the data blocks and write it on another data blocks on the other Partition, and makes a new Pointer on the new Partition!!
and then gets back and deletes the Pointer/s which was pointing to the data and deletes its Reference!!
- It doesn't delete the Inode, It just makes it free for future use!!
- That's why the Move isn't fast!

→ When you remove a file, It just make the Inode number free!!

Go to Minute 16:00 -

******

NOTE: Never overwrite a data you recently deleted!

******

→ There could be 2 files(data blocks) in different filesystem with the same Inode number!

Soft Links:-

- Reference to Reference (A shortcut to a file)!

→ If you make a soft link to 'file1' with the name 'file2', 'file2' will point to the filename and NOT its Inode number!!
- When you set 'file1' Inode' free, 'file2' will not be able to access the data-blocks of 'file1'

NOTE: After creating 'file2', It took a '**1** Inode number' to be stored on it,
- But the 'file2' real data, is a reference to the 'file1'

- The color of SoftLink is *baby-blue*, But after deleting the parent file, the color becomes '*red*'.
- You can make a Soft Link between two different filesystems!

→ When you edit the Soft Link, it will edit the source file!!
But when you delete the Soft Link, the source file still existed!!

Hard Links:-

- Another reference to the same data-blocks!

- Hard link takes the *same Inode numbers* with the one that data-blocks took!
- When you delete a reference, the other reference(s) still can work and access these data-blocks!!

→ Hard Link must be existed on the same Filesystem, because datablocks existed in one filesystem!!

→ It is a must to use the '*absolute path*' of the source and the shortcut when you make a **SoftLink**.
If you don't do this, It may works!, but still NOT RECOMMENDED or WRONG!
In Hard Link, you don't have to use the '*absolute path*', because it points to an inode number, and the inode number is unique.

→ The difference between the Soft and Hard Link:

- The Hard Link is a new reference to the same *inode number*.
- The Soft Link is a reference pointing to another reference.

→ Hard Link is not used in the Real-World scenarios nowadays!

→ Hard Link is not allowed for directory, because directory is a special file pointing to references/files.

→ Hard Link doesn't exist in Windows!!

Disks & Partitions:-

→ **MBR** takes only **512** Bytes from the space, and Partition table takes only **64** byte from the **MBR** !!

- Every Partition can store its information in **16** bytes!
- The Maximum number of Partitions on the Disk are only ***4 Primary*** Partitions!
- We can solve the problem by making the **4**th Primary Partition an *Extended Partition*, and inside it we can make Logical Partitions
- **MBR** will have information about the **4** Partitions only.
- The Extended Partition (**4**th Partition), will have a special *Partition Table* has information about the Logical Partitions inside it!
- Logical Partitions start counting from number **5**, no matter how many Primary Partitions existed!

Revise some things, but later!!

******

→ These are all devices to read CDs (**`cdrom`** - **`cdrw`** - **`dvdrom`** - **`dvdrw`** - **`sr0`**)

`sr0`: The reader of BlueRay Disks (the disks with huge spaces)

The BlueRay can read any CD with any space

- If you have more than one CD reader or writer, it will have a different number (**`sr0`**, **`sr1`**, **`sr2`** etc)
- In Linux, all those devices are accessible with '**`/dev/sr0`**'

NOTE: Linux is pointing to **`sr0`** with the CD or DVD you have connected on your machine.

-- Day 9 –

Creating and Formatting FileSystems:-

- To show the recent connected disk, you must restart the machine, Or we can use the Package **S3_Utils**
- **MBR** is called '**dos**' when it comes to 'Disk label type', in some Linux systems dos = 'disk operating system'

→ Every Partition has a different System ID, even between the normal partitions

There are a '*normal partitions*' like = **FAT**, **NTFS**, **ext2**, **ext3** .. etc.

→ There are also *non-normal* partitions to be assembled from more then one disk and make a big '*Virtual Storage Space*'

- There are **2** famous types of Virtual Disks: **RAID** and **LVM**
- These partitions are 'special', because you can assemble them from more than one disk

WARNING: when you work with '**`fdisk`**' command, you're dealing with a 'Disk' not a 'Partition'

- Be careful before you write the changes after finish using **`fdisk`** command!
- Always verify from your work, Especially when you're deleting a partition!

→ Linux by default using '**MBR schema**' for making new partitions on a RAW disk.

- Adding new Partition means that you've added a Partition Table or you've edited it.
- Kernel is reading the MBR / Partition Table, after attaching the device or booting the machine.
  (after attaching, that's because there are *auto-swappable* disks)
- After applying anything with '**`fdisk`**', you must make the Kernel re-read MBR or PT.
- We must force the Kernel to re-read the **PT**, with the command **`partprobe`**

→ ***`ext2`*** filesystem is one of the old versions being used by Linux!!

- Mostly nowadays, people use '***`ext4`***' or '***`xfs`***'

WARNING: when working with the '**`mkfs`**', we're dealing with the partition not the disk!

- When trying to make a new filesystem using '**`mkfs`**' command without selecting the type, it will be **`ext2`** by default !!

→ inode table of every partition will called '***Super blocks***'

- You can not access any file on the partition until you know what block he is pointing to
- If the Super Block deleted, how would you access files on the disk, or how'd you know their blocks!??
  That's why it is so important.
- The system creates the 'Super Block' during the formatting process
- While the importance of the inode table, the system makes copies of the super blocks and store it in Miscellaneous parts
- This happens that when any version of Super blocks is corrupted, you still can access it.
- The number of Super blocks is being created depending on the space of the partition.

→ Every inode number is representing **1** block of the data-blocks on the disk

- No matter what the block size is, **1KB**, **1MB** or even **1GB**

→ **Partition table** still existed after deleting the partitions
- To delete the Partition table or the whole **MBR** we need to write a random data on it

→ The difference between '**`/dev/random`**' and '**`/dev/urandom`**':
- That **`random`** may corrupt the input and output of the terminal screen (you should use '**`reset`**')
- **`random`** may execute non-desired commands randomly while executing it.
- And we use them to destroy the **MBR**, or to overwrite it!

→ We can use the Redirection to overwrite the disk: **`# cat /dev/random >> /dev/sdb`**
- This way is uncontrolled, and not a preferred way! Because you don't know how much data is overwritten
- And you also don't know if the **MBR** is deleted only, or the data too.

→ Use the command '**`dd`**', disk dump, to write random data, but in a selected range
- You mustn't pass the size of blocks with b or B appended, it deals with the size of block in bytes by default.
- But you should pass if it was in Megabytes

`dd if=[/dev/random] of=[/dev/sdb] bs=[1] count=[nofblocks]`
- You can make the number of blocks **512**, to delete the **MBR**

→ When trying to create a file using random data like using **`/dev/random`** or **`/dev/urandom`**
- the file will be created but with a smaller space than we thought
- If we tried to make a file with **100Mbs** and every block is **1Mb**, the file will be at almost **8Kbs**
- This happens because, the 'random/urandom' write a random data that its output will be **100Mbs** but not its size when storing it.
- 'random/urandom' writes random data with every character used in any human language!!
- That's why it is difficult to store with Normal Encoding extension

→ Like the difference between **ANSI** Encoding and **UTF-8** Encoding
- **UTF-8** needs more space to store the data inside it, but **ANSI** not.
- Because **UTF-8** can read the **ASCII** code of any character of any human language!
- That's why the process of creating the file isn't failed! but it doesn't work like we hope!!

→ To solve this problem, we use another file '**`/dev/zero`**' which is filling these blocks with zero value.

NOTE: **Format** = **Zero Fill**

→ **MBR** : **Master Boot Record**, takes **512** bytes

1 → Partition Table takes **64** byte
2 → Boot Loader takes **446** bytes
3 → Magic number takes **2** bytes
- Magic number is like check sum for the rest of **MBR**, and used in data recovery

REMEMBER: There are Super blocks still existed in the Partition, even after deleting it
- So, when we re-create a new filesystem, with the **`mkfs`** command, it reads that there really was a filesystem on it!
- So, we use the option **`-f`** to force the operation

→ **`mkfs`** is used to build a Linux filesystem on a device, usually a hard disk partition
- The <device> argument is the device name (e.g. **`/dev/sda1`**, **`/dev/hda3`**), or a regular file that shall contain the Filesystem
- The <size> argument is the number of blocks to be used for the Filesystem.

→ **Mount Description:-**

- All files accessible in a Unix system are arranged in one big tree, the file hierarchy, rooted at **/**.
- These files can be spread out over several devices.
- The **mount** command serves to attach the filesystem found on some devices to the big tree.
- Conversely, the **umount** command will detach it again.

→ The standard form of the **mount** command is:

```
# mount -t type device dir
```

→ We must make the partition accessible by mounting it!

- After we un-mount the partition, the data still existed but we need to re-mount the disk to access it.
- Data on partition is gone after you format it!!

- We can get the partition info using '**dumpe2fs**', but it works only with **ext2**, **ext3**, **ext4**
- dump extended to filesystem
- **xfs** filesystem is Highly Recommended than **ext4**, because **xfs** is so fast rather than ext4 may take a bunch of minutes -- even if the hard was SSD type !!

→ If the Electricity off, you might have partial data corruption

- And then to solve this problem you need to check disk....

→ You must unmount the partition before you check the disk.

- If you didn't unmount, you'll have data corruption

NOTE: You must take a backup to the partition before you -force it!

- It might happens when you check, there may be a data corruption, so you need force to check the partition,
- And it gives error in a block it will offer you to fix the problem and you tell it yes.

→ Filesystem check, checks only the inode tables.

- It checks to see if the inode table is consistent, if not it will only change the entry point of the inode table

→ To take a backup of the whole partition even when you can not mount it!!

```
# dd if=/dev/sdb1 of=/sdb1-backup
```

Some Revision Notes:-

1) If you mounted a partition without setting it a filesystem, you will get an Error!
2) While working with a mounted-partition, we work with its mounted point if we were working with the files related to it But when we work with partition itself, we must use its real path with its disk like **/dev/sdb1**.
3) We can dump (get the info) for the partitions with kind of **ext2**, **ext3**, and **ext4** without any need to mount them first. But when it comes to **xfs** type, we need to mount it first before using the command (**xfs_info**) !

-- Day 10 --

Mounting:-

- Root is the parent of itself,

→ **Mounting** in Linux means, linking the partition you mount with the related drive point on disk, and make the partition accessible across that point

- Mount Point, is the point that across it you can access the partition

- Linux unmounts the root filesystem while shutdown, and while booting it searches the root filesystem and mount it automatically

- In Linux, you mount any partition under the **/** root filesystem !!

- When you mount any partition, physically, the mount point takes the space of that partition.

→ When you unmount any partition, the command '**umount**', is unmounting the mounted partitions in reversing order.

→ If you mounted **2** partitions in the same mount point, the mount point will have the space of the latest partition mounted,

- And **NOTE**:
  - **1**) If you unmounted the second partition, the space will get back to the first partition!
  - **2**) If you unmounted the first partition, the mount point will get back to the **/** root filesystem

→ Never format the Extended Partition.
- All you just need is to format the Logical Partitions inside it.

→ The standard form of '**mount**' command

```
# mount -t [type] [source path] [mount point destination]
```

- That was happening in the past, you should had to send the filesystem type, but now this happens automatically
- CentOS and RHEL **4** and **5** don't have that auto-detect, but **6** and **7** have.

→ When un-mounting any device, you can pass the device or its mount point to the **umount** command.

ex: '**umount /dev/sdb1**' or '**umount /media/**', while '**/media/**' is the mount point of the device '**/dev/sdb1**'

→ The Problem with the mounting, is changing the place of devices:-
- When you connect your device to a machine you're using a specific slot with a specific symbol.
- But when you re-connect it in another slot, you're now having a new symbol on the disk.
- If the disk first was '**sda**', it may be after '**sdc**'.
- When booting you'll get an Error-Message!! because you're mounting '**sda1**' in '**/**'
  and '**sda2**' in '**/media/**' etc, and now there are a big problem because they're not found as '**sda**'
- The Boot will fail and break, and Error-Message says that you must enter the root password to maintain the operation!!!

→→ To overcome this problem we have 2 methods:

1) We use '**blkid**' command:
- When you format a partition, it gets a random UUID automatically
- UUID length is too long, and its generation contains letters and characters!
- That's why You will never see **2** partitions have the same UUID :)
- So we mounting/unmounting the device using its UUID, ex: '**mount UUID="" /media/**', or: '**umount UUID=""**'

→ UUID is the BEST and PREFERRED method!

2) Using a Human readable name:
- We use the command '**`e2label [label]`**', or '**`xfs_admin -L [label]`**'
- These two commands give the partition an attribute called LABEL with a name we set to the partition.
- Now we mount/unmount with the LABEL, ex: '**`umount LABEL=P4rtition`**', or '**`mount LABEL=SDBPart0 /media/`**'
→ This method is NOT preferred!

→ Typically in Data-Centers, we make an identical setup
- Say that you have about **100** VMs or servers, The best way for all of these **100** servers is to have a standard-setup
(**`/`** takes **10GB**, **`/var/`** takes **5GB**, **`/Oracle/`** takes **100GB** and is the Database)
- Rather than every machine have a special-setup

→ If you have a disk on the **server1** and the Database '**`/dev/sda3`**' is named '**`Oracle`**'
- when you connect that disk to **server2**, '**`/dev/sdb3`**' which has a Database named '**`Oracle`**' too
- When you try to mount which one will you mount, both have the same LABEL!??
- That's why the UUID method is PREFRRED :)

→ When you reboot the system, you'll lose the mounting you have had before !!
→ To overcome this problem,
- Everything you do on a Linux system without setting it on a Configuration file, doesn't have any value unless you save it!!
- To make the system auto mount the Disks you connect after booting:
- You must save these changes in a configuration file in the '**`/etc/fstab`**'
→ The Syntax of the **FSTAB** is:
`[device] [mount point] [type] [mount options] [dump order] [FS check order]`

Mount Options:
- **`rw`** = read write, **`ro`** = read only, **`exec`** = execute any binary or script on the mount point
- **`no exec`** = do not execute any binary or script on the mount point
- **`defaults`** = (By Default it executes the binaries or scripts on the mount point) and also read write!!

FS Check order:
- When force shutdown or unproperly shutdown happens, we use the FS check order to tell which partition will be checked first:
- its valid value is (**0** : **9**)
- **0** means don't check anything.
- 1 - **9** means the order of check you choose to the partitions.
- If there are **2** partitions that have the same order, it will check the first one in the FSTAB file.

Dump Order:
- "Dump" is used to achieve the meaning of "Backup", still used but no more like in the past
- You can configure the system to take the dump or the backup of the partitions while booting
- **0** means don't dump anything
- **1** - **9** means the order of dump you choose to the partitions

<u>BE CAREFUL:</u> When you editing or configuring the '**`/etc/fstab`**' file.
- You might forget anything or make mistakes in the FSTAB file:
- It will warn you that there is a problem while booting the system!
- In RHEL **6** or before, it will not give you a **1 min 30 sec** timeout before it needs the root password
- Only in RHEL or CentOS **7**, it give you the timeout and then asks for the root password to maintain the problem
- It will then take the root password and then you can edit the FSTAB file

→ To avoid this problem: We have **2** options:-
1) We use the command '**`mount -a`**', it will mount everything in the FSTAB, and will warn you if there is something wrong!
2) We can '**`cat /etc/mtab`**' and take the line of command related to the partition we need to mount
and paste it in the '**`/etc/fstab`**', that line is as same as the syntax of the FSTAB

- Before you delete any partition, be sure it doesn't exist in the '**`/etc/fstab`**' file, NOT MOUNTED
and delete it from the file and then '**`mount -a`**' and delete that partition

→→→ Mistake I made and Mr. Mustafa in the course:
- NEVER delete a Mounted Partition!! Unmount it first!!
- To write a **`mkfs`** (To make a filesystem), you must unmount the partition first.
→ Never Fucking use the (**`fdisk`** / **`cfdisk`**) command without selecting the disk _!_
You may ended up deleting the entire system :(

Archiving and Compression:-

→ There are **2** common packages to compress files:
- **gzip** → fast
- **bzip2** → higher compression ratio

→ You cannot read a compressed file on any Operating System.
- What happens in Windows with WinRAR: is that it decompress it in the temporary files.

→ Video and Audio files aren't compressed in these ways like the text files. It's completely different.

→ Difference between Compression and Archiving:-
- Compression is used to reduce the size of the files.
- Archiving is used to link or assemble many files in one file only, without reducing the size of these files.
- *Red color* means that the file is archived or compressed.

→ To compress a directory and archive it at the same time we use the option (**`-z`**) or (**`-j`**) appended to **`cf`**/**`xf`**
- ex: '**`tar cvfz`**' or '**`tar xvfz`**' and '**`tar cvfj`**' or '**`tar xvfj`**'
- While '**`z`** is **`gzip`**' package, and '**`j`** is **`bzip2`**' package
- If we used the **`gzip`** or **`bzip2`** after archiving, the extension is appended as '**`file.tar.gz`**' and '**`file.tar.bz2`**'.

→ '**`tar`**' is one of the few commands that its options don't have **`-`** sign before it.
- It may works also with the **`-`** sign, but sometimes it may gives another meaning.
- REMEMBER to add the **`.tar`** file name after the options of the command, ex: '**`tar cf files.tar file1 file2`**'
- We MUST add the (**`-f`**) option to the **`tar`** command when creating, appending to or extracting from **`.tar`** file

-- Day 11 --

Process Management:-

→ The Difference between the Process and the Program/Application:

- The Program is the binary existed on the system without running...
- The Process is the binary when it starts to consume the CPU, to take a space on the memory, or to allocate data on the disk.

→ Kernel controls Processes through the '**Process ID**' or the '**`PID`**'
- Every process has a '**Parent Process ID**' or '**`PPID`**'
- **`PPID`** is the ID of the Parent of a Process

→ Why we need **`PPID`**? for **2** reasons:

→ The **1st** reason is the Shared Memory:-

1) When an Application is open on Linux, like '**`Shell`**', and you opened '**`nano`**':
- Assuming there are Shared Libraries between them
- Linux here is using "Copy on Write" method,
- The best thing here is not ('allocating **`nano`** piece by piece'),
- The best thing is to Copy on Write the **`Shell`** and customize it due to the needs of **`nano`**
- and what still needed from **`nano`** is allocated then!!

2) Every running-process on Linux has Memory pieces called '**Pages**' or '**Memory Pages**'
- Most Apps on Linux is using 'Shared Libraries'
- If **`Shell`** and **`nano`** were written in **C** Language, **`Shell`** is written with **20** Libraries and **`nano`** **12**
- There could be some shared libraries or Memory Pages between them!!
- So the Best Practice here is to make a '*Common/Shared Memory Space*' and make both of them pointing to it!
  Rather than making a different Memory Space to each one of them
- For example, If **`nano`** wants to write on the Shared Memory, Linux will give a new Space to **`nano`**, to write what it needs in.

→ So, instead of making a special memory space for each Application, Linux solves it with the shared library method
- If one of the programs wanted to write, edit or delete, it will get its special space

→ So the **2nd** reason to know the **`PPID`** is:
- If **`nano`** is not responding, you can terminate it, but if it doesn't responding too???
- You can go then and terminate its Parent Process, as **`nano`** was built on the memory of **`Shell`**.

→ The **1st** Process running on the Linux System is '**`systemd`**' and takes **`PID`** '**1**'.
  This only in RHEL7 or CentOS7. (or any RHEL later version like RHEL8, RHEL9 and in **2024** we have now RHEL10)
→ The first **1st** Process running on the Linux System is '**`init`**' and takes **`PID`** '**1**'.
  This in RHEL5/6 and CentOS5/6 or above.

→ The job of '**`systemd`**' is calling the Processes and Services on the System.

→ If you '**`ps aux`**' and the **`tty`** is '**?**', this means the process is working in background

→ **`kill`** command is sending a 'Signal' to the process saying '*stop!*'

- Use '**`kill -l`**' to list all the Kill Signals
- The default signal used with **`kill`** is 'SIGTERM', signal terminate number '**`kill -15`**'
- SIGTERM is so polite, but if the process doesn't respond we can use SIGKILL number '**`kill -9`**'
- SIGHUP makes the process re-read or re-load the configuration file and number '**`kill -1`**'
- Using '**`kill -9`**' directly will cause Disastrous results!!
- **`kill`**, which is '**`kill -15`**' by default, is asking the process to terminate, but '**`kill -9`**' forcing it to
- So if the process was writing data on the disk and you force it to stop, you'll have data loss!!

→ Kernel itself is pointing to '**`systemd`**' and '**`kthreadd`**', that's why it's **`PPID`** is = **0**;

- Kernel is the Parent of '**`systemd`**' and '**`kthreadd`**' processes, as the Kernel starts them.
- '**`systemd`**' or '**`init`**' is the Parent of all processes

→ Try to avoid killing the process with its name, just use the **PID** of it

- Using the '**`pkill`**' command can make Disastrous results, if you are running instances of the process
- The '**`pkill`**' will terminate all opened processes with the same name, because you killed them by their uni-name.

-- Day 12 --

Process Management Cont:-

Process working in foreground is a normal process

Process working in the background is called '**daemon**', or '**service**' like in Windows OS.

→ The Parent of daemon process is the process started it,

- If you started it from the GUI, the Parent will be **GDM**, GNOME Display Manager (Graphical Interface).
- If you started it in booting by the call of **`systemd`**, its Parent will be **`systemd`**

→ If a daemon process working in the **background**, and you terminated its Parent, Its Parent automatically moves to be '**`systemd`**'

- Example:- If you started the '**`sshd`**' service from the terminal in the GUI, and you terminated the terminal
- The '**`sshd`**' won't terminated, because the process control moves to '**`systemd`**'

→ **Service**: is the process waiting request from the user to reply with his needs, to give him a specific order or to serve a specific job to him. This is in common.

- Services typically listen to a port.
- Services could also work locally on the machine, Un-nesessarily to work online.

→ Any application can work in foreground or background as a daemon.

→ '**nice value**' is the value controlling the priority of running-processes

- Its range is from [**-20** : **19**].
- Its default value is = **0**, means normal priority
- The highest priority is = **-20**, and the lowest priority = **19**;

→ Any normal user can give his process a lower priority, but can't give it higher priority!!

- The user can't even get it back on its previous priority after he gives it lower priority!
- If you try to give the process a higher priority, access will be denied !!

→REMEMBER: **`root`** can do anything :D

→ Zombie process is a running process consuming resources but not working in the real.

→ You can start the service/process with a specific priority,

- And you can also change the priority of real-time running process.

→ Typically, when you take back-up for the Database, you need to pause the Database temporarily till you finish.

- If you want to take a back-up and need it fast, you must give the process the highest priority.
- As the process can consume the machine resources as needed

`ctrl + C` terminates the process with signal **15** (**`SIGTERM`**)

`ctrl + Z` interrupts the process with signal **2** (**`SIGINT`**)

`Q` quit the opened session with signal **3** (**`SIGQUIT`**)

Searching and Locating Files and Directories:-

→ We can search commands with '**`whatis`**' and '**`whereis`**' commands.

→ To search files we have '**`locate`**' and '**`find`**'.

→ The problem with '**`locate`**' is that it depends on a Database when it searches for files.

- Linux System while working is trying to track what files/data have been added or deleted to/from the Filesystems.
- This update happens everyday, about midnight.
- We can force the Database to synchronizing with '**`updatedb`**' command, as it depends on a Database, the locate is so fast.

→ '**`find`**' is a Utility like '**`locate`**'

- Actually it is an updated-locate, that can search in the real-time not in the database.
- NOTE: **`find`** command searches the name only, but we can search and get results and variation about the name just by appending asterisk **`*`** to the last of the filename.
- '**`find`**' by default searches in the **`pwd`**, Present Working Directory.

→ To make a file with a name of two words or more, we write the name between double quotes "file name"

- Or we can use the backslash or backspace '**`\`**' = escape sequence after between the words of the name.
- ex: **`# touch new\ file\ name\`** >>>>> **`# ls`** >>>>> **`# new file name`**

→ We can take the result from the '**find**' command and put it in a special file:

- Example: **# find /etc/ -iname network -exec cp {} ./work/Day12 \;**

- Where **-exec** means execute another command and **** means don't change anything and take it as it was.

→ NOTE: '**-exec**' command needs semi-colon '**;**' after it, as it link between two commands

- That's why we've added '****' or escape sequence between them to ignore it.

- You can use any command with '-exec' command, like **cp**, **mv** or **rm** .. etc.

- You can use more than one pipe **|**, in the same time, ex: **# ls | grep -i file | less**

-- Day 13 --

VIM Editor:-

→ Vim is an Improved Advanced Text Editor, Enhanced Vi.

Vim is an advanced 'Vi' = 'Visual Editor'

vi was like **nano**

→ Vim has **3** main modes:-

1) *Command Mode*:- Since you open a file you are in the Command Mode!
- Allows you to only read the file
- You must click '**Esc**' to go to the Command Mode

2) *Insert Mode*:- To be able to edit the file

- You must click '**insert**' or '**I**' to enter this mode.

3) *Execution Mode*:- To exit or to save the file

- Click '**:**' to go to the Execution Mode
- write '**w**' to save and write
- write '**q**' to quit
- write '**wq**' to save and quit
- write '**q!**' to quit without saving

→ While you inside Vim, these are some buttons to reduce your time:

Button:	Its Job	Example:
Esc	Enter the Command Mode	
insert	Enter the Insert Mode	insert/i
:	Enter the Execution Mode	
w	Save and write edits	
q	To quit	
wq	Save and quit	
q!	Quit without saving	
n	For the nest result	
N	For the nest result	
l	Move the cursor to the right	
h	Move the cursor to the left	
j	Move the cursor to the down	
k	Move the cursor to the up	
o	Make new line under the cursor and go Insert Mode	
O	Make new line above the cursor and go Insert Mode	
set number	Write in the Exec mode to see the number of lines	

Button:	Its Job	Example:
gg	Go to the begin of the file	
G	Go o the end of the file	
d$	Delete from the cursor to end of the file	
d0	Delete from the cursor to begin of the file	
yy	copy the line	
nyy	copy more than line, n for lines	5yy
dd	To cut the line	
ndd	To cut the number of lines	7dd
p	To paste	
u	To Undo	
dw	Delete word from the cursor to end of it	
dl	Delete letter where the cursor	
x,yd	To delete lines from x to y	7,14d
.,$d	Delete from cursor to end, while . means the cursor and $ means the end"	
.	Redo the last command you did	

Button:	Some Tips and Tricks:-	Example:
`%s/word`	Search a word and replace it, go to Exec Mode and write '%s/sync/rep' '%s' to substitute /word/replaced_word	`%s/sy/re`
	NOTE: This change happens only to the 1st result existed on the same line. If the results are split into many lines, without repetition, all lines will be changed	
	To replace all the substituted words, go to Execution Mode and add '/g'	
	Example: '`%s /sync/replicate/g`', while `g` means global	
`r filename`	To read and copy the content of the file to the cursor place, from exec mode	`r file1`
`w filename`	To save the content to a new file, from Execution Mode	`w file2`
`w >> filename`	To append the content to another file, from Execution Mode	`W >> file3`
`syntax on`	To highlight the text	
`syntax off`	To un-highlight the text	
`![command]`	To execute a command in hurry, from Execution Mode	`!date`
`.![command]`	To add the result of a command to the cursor place or to the file, from Execution Mode	`.!cal`
`r ![command]`		

End of the first 1st Track ();

-- Day 13 –

Boot Loader:-

The process of booting the Linux Machine consists of **2** stages:

1) Before the Operating system booting
2) After the Operating system booting

BIOS: stands for Basic Input/Output System

→ After clicking the Power button, the BIOS checks the hardware components
1- First, it makes **POST**, power on-self test, to check the minimum hardware like CPU, RAM, Hard Disk and machine capabilities
2- Then, BIOS detects the Bootable devices and compare it with its settings, like the **1st** and **2nd** boot device
And then boots from these devices.
3- After that, **BIOS** loads the **IPL**, Initial Program Loader, its job is to execute the **BootLoader**
4- **IPL** then load/start the **BootLoader**
5- **Boot Loader** loads the Kernel
6- And then the Kernel starts the Process (**`systemd`** / **`init`**)
7- **`systemd`** then calls/starts the services/daemons.

→ Our Boot Loader in Linux is called **GRUB**, Grand Unified Boot Loader
- Boot Loader is about **446** bytes stored in the MBR

Boot Loader consists of **2** stages:
1) 1st stage: The real Boot Loader, stored in MBR
2) 2nd stage: The files related to it, stored on the disk, in **`/boot`**

The **1st** stage' job is to call the **2nd** stage ☺
This isn't a shortage of Boot Loader, but the MBR !!

→ **GRUB** is so smart, and can detect other Operating Systems like Windows, Solaris, Mac and can load them together.
→ Windows has **NTLDR**, NT Loader, it can't detect any other operating system.

→ If you have a Machine with RHEL and Windows installed
- If you got a virus on Windows, and you re-installed it, the **NTLDR** will over-write the **1st** stage of the Boot Loader in the MBR !!
- **NTLDR** doesn't care about other operating systems
- But Note that the **2nd** stage of the Boot Loader still existed with its files on the RHEL, don't try to re-install the full RHEL
- Just re-install the **1st** stage of the Boot Loader, this will overwrite the **NTLDR** and install the **1st** stage
- And the **1st** stage of the **GRUB**, will notice that you have Windows and Linux and both of them will work together again !!

NOTE: when you install Windows never try to format the MBR!

→ The configuration file of the **GRUB** is in the **`/etc`** as a soft link to itself in the **`/boot/grub2`**
- That's why we do as the rule of "Every configuration file must be in the **`/etc`**"

KERNEL :-

Kernel in Linux consists of **2** parts:
1- **`Kernel`** itself
2- **`initramfs`** (**`initrd`** in RHEL 6 and above!) initial ram drive = **initrd**

→ Kernel is splitted into **2** parts because the Kernel developers make the **Core** in the kernel file,
- But the modules and drivers in the **`initrmafs`**. If you wanna access the WiFi, network, tape device, the disk or any other thing.

→ Splitting the Kernel allows you to:-
1- The **Memory Footprint** of the Kernel will be too small, to reduce the memory used by the Kernel
2- In the booting process, loading Kernel with reduced memory size like 5MBs will be faster, rather than if it was **40** MBs !
3- If you installing a driver and the installation was corrupted, you'll need to re-generate the **initramfs** or to re-build your drivers rather than re-generate the whole Kernel, because the Kernel will be untouched!

→ You will never see that when you install a driver and the installation corrupted,
- that the entire system is corrupted and you need to re-install your machine, this doesn't happen in **Linux**!
- But may happen in Windows! -- Such is Life ;D --
- Because the Kernel of Windows is just **1** file! One file containing kernel, drivers and modules and many other things!

→ If **initramfs** was corrupted, you can re-generate it with only **1** command!

→ Kernel while loading:-
- Kernel mounts the root file system **/** in **`ro`** (read only) mode and try to ensure that there are no problems in the system!
- If you had a problem, it will fail the load.
- If Everything is OK, the kernel re-mount the root file system **/** in **`rw`** (read-write) mode.
- Because if the mounting process was in the **`rw`** mode, and you have a problem, you'll end up have more than one problem!

→ Summary:
1- Kernel Loading
2- Kernel loads some of the drivers from the **`initramfs`**
3- Kernel mounts the root file system **/**
4- Kernel calls the **`systemd`** / **`init`**

→ These are the cases you need to restart your running-server:
1- If you re-enabled the SELinux if it was disabled
2- If you upgrade the Kernel
Needing to restart the server as you need to upgrade the Kernel will be unnecessary soon.
Because there is K-Patch or Kernel Patch, which allows you to upgrade the Kernel during running the machine!

-- Day 14 --

GRUB supports:
1- More than one Operating System
2- Making Security settings to it, like making password to it to prevent anyone from editing

systemd :-

→ init and systemd similarities:
- Always takes Process ID of number **1**
- Its job is to call the services to run

→ init and systemd differences:
1- When **init** calls the services to run, it calls the **1st** service and wait for it to respond (respond if failed or worked),
And then calls the **2nd** service and also wait for it to respond, and then calls the 3rd service.. etc.
If every process needs about **3** seconds to run, the **3** services needs **9** seconds to work which is a waste of time!!
On the other side, **systemd**, run the services **in parallel**, calls the services in the same time and wait for them to work!!
If every process of the **3** processes needs **3** seconds to run, the **3** processes together will need to only **3** seconds to work!!

2- **init** calls the services with an "init script", every service has its own "init script", written in Shell Script!
systemd has "Unit files", normal text files like scripts but without codes,
These "Unit files" contain only texts like: var = value; makes debugging and troubleshooting much easier than "init scripts"!

3- **systemd** is compatible with **init**, because **systemd** allows **init** to work with it in the same time
That's why if you have a service and it was written in **init** shell script, you don't need to run that service on an earlier version of RHEL and also you don't have to write a Unit file for that service to work with **systemd**!

4- **init** has a different command for every task you need to do.
systemd has **1** command gives you full control on every service you have! '**systemctl**' system control

5- We had a service called **syslog**, and was developed to be **rsyslog**, was to recording logs.
So if the system is booting in RHEL 6 and there was a service running before **syslog**, you will not be able to see its logs.
And if there was a problem you also will never be able to know what the problem exactly is with the service!
To avoid that problem with **init** in **systemd**, they wrote **journald,** also to recording the logs.
When you want to search the logs in **syslog** or **rsyslog**, you were searching any problem in the logs, you were searching the whole file without filtering!
journald has a command '**journalctl**', you can search with this command about a specific-service with filters!

6- **init** starts in Sequential Boot, but **systemd** in Parallel Boot.

→ The development between **init** and **systemd** didn't happen overnight!
- There were many implementations, like **upstart** developed by Ubuntu team pushing people to use **upstart**.
- And it was also using **Parallel Booting System**!
- After that Fedora developers wrote **systemd** and both Fedora and Ubuntu developers agreed to use **systemd** !
- **systemd** also compatible with **upstart**.

→ Apcahe service is called **hhtpd**

→ If the service status is active, this means that the service is currently working.
- enabled or disabled status tells the machine if the service will work once the machine rebooting or not!
- You will need to make it run manually after rebooting!

→ Every service we install comes with default configurations, may be it is enabled to work after booting by default or not (disabled)
- We can also change it whenever we want to.

Real World Scenario

- If you have more than one implementation for a service like **Apache** and **nginx**
- These two services can be installed on the same time on the machine.
- Butt they can't work on the same time as they listen on the same port, port **80**.
- **Apache** has many features than **nginx**, but also slower than **nginx**!

→ Mr.Mustafa stopped Apache and installed nginx because the web server was too heavy and also disabled the Apache service.
- The Genius working with him reboots the system and found that the web service wasn't running, so he turned on Apache!
- This is a خازوق !
→ To avoid this problem:
- We can **mask** the service, preventing the service from running even if someone starts it!
- After that we can test the new service we installed and if we want to get back to the service, we can **unmask** it again!

→ **`systemd`** calls the mode of GUI.
- So the Basic is CLI not the GUI
- You can't work only in GUI without the CLI
- As **`systemd`** can work in more than one mode, these modes are called **targets**.

→ **`init`** has modes also but called **run levels**.
- Windows has modes also like: **Safe Mode**, **Safe Mode with Networking**, **Recovery Mode** ..etc
- Windows has the same concept of run levels or modes.

→ `init` run levels:

init run levels		
Run Level	**Job**	**Examples**
0	Power-off / halt	Shutdown the Machine
1	Single User Mode	Emergency Mode, Used for Troubleshooting, Only root has access
2	Multi User Mode, without GUI and NFS	No Network but Multitasking support is present
3	CLI, only TTYs, also Multi User Mode	Servers works on this level, Network is present but without GUI
4	Unused, was reserved for future used	Similar to init3, but reserved for other purposes in research
5	Multi User Mode with GUI	Used in Desktop, Network and GUI is resent and also sound
6	Reboot	System restart
s, S, m or M All same	Tells init command to enter the Maintenance Mode	When the system enters Maintenance Mode from another run level, Only the system console is used as the terminal

→ **NFS**: is a service used to share files between Linux and Linux or Linux and Unix Machines .. etc.
- Usually, Servers were working in the **3rd** run level of **`init`** and the Desktop Machines in the **5th** run level.
→ **GUI** is the most thing consuming resources.
- The **1st** run level, Single User Mode, was for troubleshooting, it allows you to get **root** privileges without asking for the password.

→ `systemd` targets:

systemd targets		
SysV Runlevels	**Target Unit**	**Description**
0	**`runlevel0.target, poweroff.target`**	Shut down and power off the system
1	**`runlevel1.target, rescue.target`**	Set up a rescue shell, equivalent to Single-User Mode Allows you to to repair the system and troubleshooting
2	**`runlevel2.target, multi-user.target`**	Set up a non-graphical Multi-User Mode, without GUI or NFS
3	**`runlevel3.target, multi-user.target`**	Set up a non-graphical Multi-User Mode, without GUI or NFS
4	**`runlevel4.target, multi-user.target`**	Set up a non-graphical Multi-User Mode
5	**`runlevel5.target, graphical.target`**	Set up a graphical Multi-User Mode with GUI, Network and Sound
6	**`runlevel6.target, reboot.target`**	Shutdown and reboot the system

The **default target** is the target started when you boot the machine.

→ The result of the '**runlevel**' command is like "**N 3**", which **N** is the ex-runlevel and **3** is the current run level.
- But N here means "Not available", as it doesn't know what the run level before.
- You can change the current target while running the machine.

→ The difference between '**isolate**' and '**set-default**' is that **set-default** changes after rebooting,
- But **isolate** changes the running session.

init command is still exited on Linux, just for compatibility.

→ The main source to most of the files of the **systemd** is in **/usr/lib/systemd/**

→ The Linux system makes a soft link of the services in **/etc**
- To enable the service, just make a soft link of it in the **/etc**
- To disable the service, just delete the soft link.

→ **init** starts in Sequential Boot, but **systemd** in Parallel Boot.

→ One of the most important features in **systemd** is: Dependency-Based Boot
- That allows to **only** run a service after a specific service.
- And tells that a service doesn't run **only if** a specific service run before it.
- Example: To don't run Apache while the Network is down, to prevent consuming the machine resources

→ If the service state was "**static**", this means that this isn't running in the background, but the system used it and it doesn't in need in the real time.

→ The difference between the '**restart**' and '**reload**' to the service when using **systemctl** command:
- Is that restart, turn off the service and run it again.
- But reload, makes the service re-read the configuration file and apply it.

-- Day 15 --

GRUB Boot Loader:-

→ Menu-Entry in the GRUB config file, is a selectable line to allows you to choose which Kernel version you choose to boot from, Or to choose another Operating System if you have.
- You can install more than one Kernel version in Linux system.
- But when booting you can choose only one version to boot from it.
- This also allows you to test the new version of Kernel you recently installed.
- Every Kernel version has a version of **initramfs**.

Remember:-
Kernel naming the disks with its physical type, like giving **sda, sdb** for the SATA, SSD and SCSI disks
And **hda, hdb** for ATA and Parallel ATA, and **vd** for virtual disks like on the virtual machines.
All of these names are human readable.

→ Kernel is which giving the names of partitions like "sda1, sda2, sdb1, sdb2, hda1, vd etc", but GRUB is so low level than Kernel!
- That's why the GRUB is reading all Disks with the name "**hd**" but without lettering it, it numbers it with numbers like "**hd0**, **hd1**, **hd2** et cetera",
- This happens because the GRUB can't read the name of Disks because it loads before the Kernel.
- And also, the GRUB identifies the Partitions with MBR, with the name "**msdos1**, **msdos2**, **msdos3**"
- So the complete name for **sda1** → **hd0,msdos1** and **sda2** → **hd0,msdos2** and **sdb3** → **hd1,msdos3**. For the GPT (**hd0,gpt1**)

→ When GRUB is installed, it detects the disks on the machine and detects their type, rather it was **dos=MBR** or **GPT**
- And then the GRUB auto-detect the Operating Systems installed on the disks and which partition where it's installed.

The Difference between the MBR and GPT:-

1- MBR is written at the first of the disk, but GPT is written at the first and at the end of the disk.
2- MBR can have only **4** primary partitions, but GPT can have **128** partitions!
3- MBR has a limit space reach to **2TB**, but the GPT can reach to **9 ZB**!!

NOTE: If you make the disk MBR and you wanted to turn it to GPT you will have Data Loss! And you will need to make the disk raw first and then install the GPT. And vice versa is right, if it was GPT and you need it MBR!!

→ Some Programs take a backup copy of the MBR and put it in the Memory and then delete the MBR and install GPT, and after that restore the begin and end of every partition and puts it in the GPT! It works with some programs but not all!!

→ You may not see any program that converts the GPT to MBR, because MBR is **32**-bit Architecture and GPT is **64**-bit Architecture.
- And you can not put the **64** bit in the **32** bit 😊 but vice versa is right!
- MBR = msdos = LBA (Logical Block Addressing)
- GPT = GUID Partitioning = EFI = uEFI (Universal Extensible Firmware Interface)

→ UUID of the partition changes in one case, if you format that Partition because it is generated randomly!

→ In the Menu-Entry, the system searches the root file system with the name of it "**hd0,msdos1**" and its UUID
- If it isn't found ☹ It then searches with UUID only, as it is a unique key to the partition!

→ In the GRUB cfg file, you can delete "rhgb quiet", to not see the graphical boot and to not be quiet means to be able to see what happens in the background while booting and what the services starting.
- Being able to see the boot log can help you in many situations, like if the Apache fail to run, you can see in the boot log what is the problem with the Apache service!

NOTE: You can re-generate the GRUB configuration file with the command '**`grub2-mkconfig`**'
- Everything you've edited before will get back to its main source.
- Everything you've edited will be overwritten immediately after re-generating the GRUB.

→ Sometimes you need to make these edits still, even after re-generating the GRUB configuration file.
- To make this, you need to edit the config file of the RUB in the "**`/etc/sysconfig/grub`**"
- After editing the file, you **must** re-generate the GRUB config!

→ GRUB start counting the Menu Entries from number **0**.

→ The cases to make re-generate the GRUB:
1- You've installed a new version of Kernel.
2- You've installed/updated the package of the version of **GRUB2**.
3- You've edited this file manually to edit its settings.

SELinux:-

- SELinux classifies the files on the system and gives every file a LABEL.
- LABEL's job is to give every file a specific access.

→ If you forget the **`root`** password and you get into the Recovery Mode to re-set it, you'll edit the "**`/etc/shadow`**" you'll change the LABEL of the file or maybe you'll delete it!!
- If the LABEL edited, and Kernel is booting and look to its LABEL, **SELinux** will prevent the Kernel, that this LABEL isn't recognized.
- You may have changed **`root`** password, but **`root`** can't login as the Kernel can't read the file **`/etc/shadow`** because of SELinux
- SELinux gives every file a Label, and if the Label doesn't match the **Policy**, the Kernel will not be able to do anything to this.
→ And the **Policy** says that **`/etc/shadow`** must be with label **shadow_t**

NOTE: Recovery Mode in Linux does not support the **SELinux**
➔ After changing the root password, you must make the system re-set the Label to its origin!!

Re-setting root password:

- We can edit the Menu-Entry from the machine while booting by clicking "**e**" button.
- We add "**rd.break**" in the last of the line before loading the **initramfs**,
- which will make the kernel fails the boot after it loads the **initramfs** file and after this you'll get root access on the machine!

→ Kernel has a Mini-Version of a File System Looks Like **/** , but not the main one,
- This version is the version Mounted in the memory when you enter the Recovery Mode!
- And your real partition of the disk **sda1** will be mounted in **/sysroot**

→ **REMEMBER:** Kernel is loading the root FS as **read-only** mode! To switch to the real File System, you need first to mount it in the **read-write** mode!!
→ To change the mount mode we use the command "**mount -o remount,rw /sysroot/**"
- And then we change the **root** filesystem to the **/sysroot/** with the command '**chroot /sysroot/**' !
- After that we change the password with '**passwd**' command and type the new password!
→ If we take a look to the Label of '**/etc/shadow**' we will set it as '**?**' so we must change it.
- To change we need to make it manually and this is the hard way.
- Or we can do it automatically by making a file '**touch /.autorelabel**'
- This file will search all files and try to find what the Policies of these files must be and apply or re-label these files due to the Policy!
- If we created that file under the **/** , it ensures that SELinux checks the files and re-label it before the Kernel starts!
NOTE: this file tells the SELinux that there is a file has some changes please re-label it again due to the Policies.

→ You must make a Password for the GRUB to avoid the change of the physical change of the root password!!

→ If you forget the GRUB password too, you can attach your disk with another machine and do the same steps!
- Or you can start booting from a CD and entering the Recovery Mode!

→ You can encrypt the Hard Disk to Prevent this process because it has many risks, and someone can steal your data.

-- Day 16 –

Disable Ctrl + Alt + Delete:-

- While working in the machine, If you accidentally clicked **Alt** + **Ctrl** + **Del**, the Machine will restart!!
- This causes a disastrous problem in the Production Environment!
- Your role as a **System Admin** is to prevent these Alt + Ctrl + Del from working.

→ What causes the Ctrl + Alt + Del to reboot is the soft link found in the '**/usr/lib/systemd/system/ctrl-alt-del.target**' and is a soft link to the file '**../systemd/system/reboot.target**'.
- The file was controlling **init** to make the Ctrl Alt Del to reboot was the file '**/etc/inittab**'.
- If you want to prevent this in RHEL 5 or 6 you just comment **#** the file of the **inittab.**
→ But Now as this file is a soft link to a real **systemd target**,
- You just can **mask** it with the command '**systemctl mask ctrl-alt-del.target**'.
- Now if you clicked **Ctrl** + **Alt** + **Delete**, the system will not re-boot!!

→ We **masked** a target not a service! So, we must do '**systemctl daemon-reload**' after that to reload all the configuration files that we've edited recently!

→ After these steps, if you '**ls -l**' to the **ctrl-alt-del.target** you will see that it stills pointing to **reboot.target**!!
- But if you see the file in the '**/etc/systemd/system/**' you'll wee the file pointing to '**/dev/null**'
- As the configurations must be edited in the **/etc** not in the **/usr/lib/systemd/system/**, so the **mask** didn't edit it.
- It just take a soft link from it in the '**/etc/systemd/system/**' because what existed in the real '**/usr/lib**' can't be edited!
NOTE: We must do '**systemctl daemon-reload**' after editing the configuration files especially when editing the targets.

→ What the **masking** really do, is that it makes the file as a soft link to the Special File '**`/dev/null`**'.
- Also if you want to delete a file in Secure Way, just make it pointing to the '**`/dev/null`**'!!

NOTE: **`systemd`** while working, it takes its configurations and settings from **`/etc/systemd/`** and it has the priority!
- If it is not found there, it takes it from the **`/usr/lib/systemd/`**.

➔ If you want to disable it in the RHEL6 you can go to the file '**`/etc/init/control-alt-delete.conf`**'
And inside the file, you can comment **#** the execution command!

Network Manager:-

We have **2** implementations to Network on Linux:
- Network (Old)
- Network Manager (New)

Network was so easy to configure, but if you want to configure VLAN you'll need other commands with other modules to configure
If you want to configure Bonding, you need an additional package installed first!

→ Network Manager allows you to:-
- Configure Basic TCP/IP
- Configure VLANs
- Configure Bonding / Nig Teaming
- Support advanced features like connecting to VPN connections.
- It consists of **Building Blocks**, If you want to change the VLAN implementation you don't have to change the whole code.

→ The Network Manager has a feature makes it easier to use like the **`systemd`**, which is the one command used '**`nmcli`**'.

→ Every PC has an implementation to the TCP/IP Stack will have a **LoopBack Interface** as a Virtual/Software Interface.
- The **LoopBack**'s job is to test the TCP/IP implementation, whether it works correctly or not.

→ The IP **127.0.0.1** is locally and existed on every PC.
- It allows you to run local services on your machine like: MySQL, Apache or NginX.

The second Part of this Day was about Networking and IPs and Subnet mask…

-- Day 17 --

The Default Gateway is the gateway that connect you to the Internet by default!
But the normal* Gateway is the way to a specific network.

DNS stands for **Domain Name Server**.

Default-gateway typically used with Layer2 Switches.
Default-route typically used with Layer3 Switches, Routers and Firewalls.

This because the L2 Switches can't do Layer3 Packet Forwarding or (Routing), That's why it is called Default-gateway!
And L3 Switches, Routers and Firewalls can do Packet Forwarding or (Routing), so they called Default-route!
Network Manager makes you work with **Profiles**.
You can make a **Profile** to every connection of your connections and then active the **Profile** you want due to your needs!

You make a Profile only one time and active it when need it.
You can not active **2** Profiles at the same time.!

When adding a connection with the Network Manager, if you didn't select the type, it will choose **Ethernet** automatically.
autoconnect option to choose yes or no to bring this connection up automatically, and when you re-boot to connect automatically or no, If you don't make it yes you'll need to make it manually later!!

When adding a connection and you didn't the way of Linking, it will automatically take an **IP** from the **DHCP** by default!

DHCP stands for **Dynamic Host Configuration Protocol**.
It's job to give any device connected on the Network an IP address from the range of the IPs.

If you modified a connection, the settings of it will last as it.
You need to re-activate your connection to apply the new modifications you have added. Make down and up to the connection!

**You can make more than one profile on the same Network Device!

-- Day 18 --