**Red Hat Certification System Administrator**

# To learn

What is PXE and kickStart in RHEL

sudo su -

activer quota avec fstab linux

/etc/exports File

enlever le no root squash

ldap et nis

What is home\_nfs

What is auto fs

What is slurm

What dos /net contain

What is ssh -J

multipath -l

Difference between df and du

lsblk

vgdisplay -v

What is mofed

What is kickstart

# Linux Architecture

The kernel is linked directly to the hardware components. The shell is a higher level than the kernel and we have the highest level that is the closest to the user which is the user interface.

In linux, everything is a file. It is known as flat file so for example in linux a directory is literally a file with the type ‘d’. So if a virus is spread, it won’t be able to spread across files inside of the directory.

# Linux overvirew

In linux, everything is a file. Linux is known as flat file so for example a directory is literally a file with the type ‘d’ which means that if a virus is spread, It won’t be able to spread across files inside

# Basic directories present in Linux OS

- Under the ‘/’ directory which is the root we have multiple directories like home directory, root directory for the root, dev, etc, bin etc…

- /dev : Used to store device nodes for physical hardware and virtual devices. The Linux kernel communicates with these devices through corresponding device nodes located here. These device nodes are automatically created and deleted by the udevd service (a Linux service for dynamic device management)

There are two types of device files: character device files, and block device files. The kernel accesses devices using one of these files or both.

* Character devices are accessed serially with streams of bits transferred during kernel and device communication. Examples of such devices are console, serial printers, mice, keyboards, terminals, etc.
* Block devices are accessed in a parallel fashion with data exchanged in blocks (parallel) during kernel and device communication. Data on block devices is accessed randomly. Examples of block devices are hard disk drives, optical drives, parallel printers, etc.

- /etc : Contains config files. Does not contain binary files like scripts

- /bin : Contains essential user programs and commands like cat, cd, etc…

- /sbin : Like bin but contains also commands intended to be executed by root administrator

- /home : Contain all user profiles

- /lib : Contains libraries needed by /bin

- /mnt : It’s where system admins mounted temporary file systems while using them

- /media : Contains subdirectories where removable media device inserted into the computer are mounted

- /opt : Contains subdirectories for optional software packages. For example, if we install chrome browser, it will be stored here

- /proc : Contains special files that represent system and process info. Used to maintain information about the current state of the running kernel. The contents in /proc are created in memory at system boot time, updated during runtime, and destroyed at system shutdown.

- /run : This virtual file system is a repository of data for processes running on the system

- /run/media : One of its subdirectories, /run/media, is also used to automatically mount external file systems such as those that are on optical (CD and DVD) and flash USB. The contents of this file system are automatically deleted at system shutdown.

- /srv : Contains data for services provided by the system. So app data can be stored here

- /sys : Virtual file system which stores info about devices connected to the system. This information is used by the kernel to load necessary support for the devices, create device nodes in /dev, and configure the devices. This file system is auto-maintained as well.

- /usr : Contains apps and files used by users as opposed to apps and files used by the system

- /usr/local : This directory serves as a system administrator repository for storing commands and tools downloaded from the web, developed in-house, or obtained elsewhere. These commands and tools are not generally included with the original Linux distribution. In particular, /usr/local/bin holds executables, /usr/local/etc holds configuration files, and /usr/local/lib and /usr/local/lib64 holds library routines.

- /usr/share : This is the directory location for manual pages, documentation, sample templates, configuration files, etc., that may be shared with other Linux platforms.

- /var : Contains logs, mail, temps files… contains data that frequently changes while the system is operational.

-/var/spool: Directories that hold print jobs, cron jobs, mail messages, and other queued items before being sent out to their intended destinations are located here.

- /var/tmp: Large temporary files or temporary files that need to exist for longer periods of time than what is typically allowed in another temporary directory, /tmp, are stored here. These files survive system reboots and are automatically deleted if they are not accessed or modified for a period of 30 days.

- /tmp : Contains temporary files. Accessible by root and user. When reboot happens, data in /tmp will be deleted. The default number of days files in /tmp are kept before they are automatically deleted if not accessed or modified is 10 days.

# Difference between echo and printf

Echo takes the input and transform to stdout so to output (prints on the screen) but with printf we can have more utilities.

# Connect to BMC

ipmitool -H mexico102-bmc -U ADMIN -P ADMIN -I lanplus sol activate

# file command

**file** anaconda-ks.cfg : show us the type of file (Ascii)

# stat command

**stat** anaconda-ks.cfg: The two commands (See command **file**) report the file type differently. The first command returns the specific type of data that the file contains (ASCII text), and the latter simply states that it is a regular file. But the **stat** command show us the nb of blocks for the file.

# Which three Linux utilities can be used to determine a file’s type?

The stat, file, and ls commands can be used to determine a file’s type.

# udevd service

Which Linux service creates device nodes at system startup? The udevd service

# Sort command

We can use this command to sort a text file by alphabeletical order or by…

# What command would sort, in descending order, a text file containing a list of employee names?

sort -r inputfile.txt

# What command would list only the first names from a comma separated file containing only employee names where the first field contains the last name and the second field contains the first name?

cut –f 2 –d”,” employees.csv

# Which command deletes any line that contains the word “remove” from a text file called myInfo.txt?

sed ‘/remove/d’ myinfo.txt

# Dmesg command

This command is used to troubleshoot hardware problems. When we execute this command we will have a trace on when the kernel was booted and then the devices that he discovered etc… The time in seconds that we see in the beginning of each line is how much time it took for the kernel to do each itenary.

# sudo su - command

Makes me root without having to know root password.

# ss command

Socket session. In fact some people says socket others say session but it is the same thing.

**ss -an** : The ‘-a’ stands for show me all of my opened sessions (au niveau TCP par exemple)

# dhclient command

This command will turn on the machine to work as a dhcp client. Normally it should be up by default.

We can also retry while using the ‘-r’ parameter: **dhclient -r**

# Restart network service

**Systemctl restart network** : We can use this command to restart the network service so for example if we have problems with our network adapter drivers…

# Interfaces config files

The interfaces config files exists in the following directory : **/etc/sysconfig/network-scripts/ifcfg-ens33** . So here for example we took the config file of our machine interface.

There is also a file in the sysconfig which is the **/etc/sysconfig/network** . This file contains global config so a config that will apply to all network interfaces. However, the interfaces config files overwrites it .

# DNS name servers

The DNS name servers are defined in the file **/etc/resolv.conf .** This file also is the global for all interfaces config but it can be overwritten by the file **/etc/hosts**

* **/etc/resolv.conf**: Lists nameservers that are used by your host for DNS resolution. If you are using DHCP, this file is automatically populated with DNS record issued by DHCP server.
* **/etc/hosts/**: It is just a static lookup method for resolution.

# nmcli

Network manager cli is used to facilitate the work so it knows which files to modify etc…

**nmcli device status :** Will show us interfaces with its status

**nmcli device show ens33**

**nmcli connection edit ens33 :** Here he will have a shell to configure the interface for example we can do **set connection.autoconnect yes** or for example **set** **ipv4.method manual** and then **set ipv4.addr 192.168.0.2/24**

We should not forget to save otherwise its gonna be erased after reboot

**Save persistent**

And finally we reload the connection

**nmcli connection reload**

# iftop command

**iftop -i eno2** : Voir le traffic qui passe sur une interface et le destinataire.

# ip route command

To see the ip of our default gateway we can use the ip route command and here we will see an entry saying : Default via 10.0.01 for example here we would know that the ip of our default gateway is 10.0.0.1

**ip route get** 10.1 : Show us route to this network

# Traceroute and tracepath

**Tracepath** [**www.google.com**](http://www.google.com): Tracepath is the newer version of traceroute and it is more advanced

# Ip-helper-adress on switch L3

**ip-helper address** : Utilisé sur les switchs L3 et routeur pour empêcher de mettre en place un serveur DHCP pour chaque subnet mais utilisé à sa place un seul relay DHCP qui lui va transmettre les requêtes DHCP entre les différents subnets.

# tcpdump command

**tcpdump -i ens33 > data.txt** : The tcpdump will capture paquets so here we are telling it to capture the packets on the ens33 interface and we are sending the data to a text file.

# sssd service

This service is the LDAP.

# Configuring LDAP on RHEL 8 (Follow the link)

[*Chapter 2. Understanding SSSD and its benefits Red Hat Enterprise Linux 8 | Red Hat Customer Portal*](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/configuring_authentication_and_authorization_in_rhel/understanding-sssd-and-its-benefits_configuring-authentication-and-authorization-in-rhel)

# netcat command

To scan ports like port 80

# Rpm and Deb

Rpm packages are used by redhat and Debian packages are used by Debian

# Apt command (Debian only/not compatible with red hat)

Apt is the new command that combine apt-get and apt-cache.

**Apt install perl** : So here when we use apt to install a software and not install it from a website, we are using repositories to do so, so apt will look in **/etc/apt/sources.list** to find these repos. There is also a file called **/etc/apt/sources.list.d** and here normally we create several files for a categories of repo so he will have something more organized but normally we would see all repo in the **/etc/apt/sources.list**

# Apt list command and apt search command (Debian only)

**apt list apache** : This command is used to search for software packages so for example I can search for apache.

**apt search apache** : This command does a deeper search of packages so if we really don’t know a lot about the name of the package we want to install.

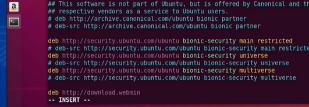
# Apt install command

By default, apt will search for packages inside of the repo but I can say to apt to look for a file that I downloaded and is not in the repo

**Apt install web\_1.5.deb** : So here we are installing it from a downloaded package from a website for example. However, with that method we will not have updates. We can sync a new repo for that package.

# Sync apt repo

So if we read th section just above we will understand that here we want to sync a repo so we will add a repo to **/etc/apt/sources.list** or we can create a new file specific for that in the **/etc/apt/sources.list.d.** In that way we could update packages with apt.



I can use the command **Apt-key add jcameron-key.asc** : So here we downloaded the public key so that we are sure that our package is safe

Another important thing to do is to update the cache of apt

**Apt update**

Here it is not updating the packages but only the cache of apt.

To really update the packages I can use

**Apt upgrade**

# Update kernel

**Apt dist-upgrade** : Need reboot after upgrade.

# Difference between su and su -

**su** will switch to a specific user without executing the scripts by default but **su -** will execute the scripts by default. (See part “Type of shells”)

# When changing user contexts what benefit is there to using su - <username> instead of su <username>?

It creates a full login session that changes directories. The difference between the **su** and the hyphenated **su –** commands is the **su** command without arguments keeps almost all environment variables belonging to the original user. Contrary to this, the hyphenated **su –** command clears most environment variables.

**su -**

# What is the purpose of the sticky bit in Linux permissions?

Ensure that only file creators can delete their files. A permission bit which is set on a file or folder, thereby permitting only the owner or root user of the file or folder to modify, rename or delete the concerned directory or file

# What do setfacl and getfacl commands enable for Linux administrators?

Access to more advanced and granular permissions

# Which command can format the first partition of /dev/sdb using the XFS file system?

mkfs -t xfs /dev/sdb1

# What is the range and priority (from fastest to slowest) for setting a nice value for a Linux process?

-20 to 19

# bash history

Bash history is stored in the a hidden file in the home directory of a user, that’s why when we close and reopen a session we will always have the history of the executed commands. This file is stored in **~./bash\_history**

# Run level command

In fact, in Linux either we launch our session with GUI or without GUI. If it’s with GUI we are using the run level 5 or the binary value 5 and if without we are using the run level 3. We can visualize the history of this with the **runlevel** command.



So here we see that the runlevel has a value of 3 which means without GUI but now have the value 5 which means the contrary.

# Launch a process in the background

We can use **&** sign to make a process run in the background.

**gedit &**

# Difference between kill and killall?

kill accepts process ID numbers as an argument, and only kills one process at a time (unless you specify multiple process IDs in your command) killall allows us to kill processes by name and will end all processes that have a matching name

# lsof command

sof is a command meaning "**list open files**", which is used in many Unix-like systems to report a list of all open files and the processes that opened them.

# systemd analyse command

may be used to determine system boot-up performance statistics and retrieve other state and tracing information from the system and service manager.

# In what state of the process life cycle is a process that is a child of a process that has finished but has not been told to stop by its parent process?

Zombie State

# What really contains the /dev repo?

Each piece of hardware in the system has an associated file in the /dev directory that is used by the system to communicate with that device.

In fact, the /dev directory creates files that matches hardware nodes so for example of our PC cd rom, we will have a file called cdrome in the /dev directory. The files names could change from a distro to another. So here are hardwares that were detected by the system so for example if we remove the video card and we reboot the system we won’t see the file corresponding to this piece of hardware in the /dev directory.

We might have 2 same distro but there are some different names for th hardware pieces in the /dev repo and this is the case because there has been some overwrites that has been done. We can verify the rules file in **/etc/udev/rules.d** to see how the names are configured…

There are two types of device files: character (or raw) and block. In the example below, the ls command in field 1 of column 1 distinguishes between the two with a “c” for character and a “b” for block:

**ls -l /dev/console**: Will show us ‘c’ as device type which means character

**ls -l /dev/sd\***: Will show us ‘b’ as device type which means block

# Udevadm command

We can force udev the reuse the normal /dev names and reload its rules by using the following command : **udevadm control --reload-rules**

**udevadm trigger**

So here without rebooting the system we will reload the names.

# See all services

**systemctl list-units --type=service**: This command will show us all services running on our system

Ou bien

**systemctl --type=service**

# Check status of services

**systemctl** **status** sshd : This command will check the status of the service sshd which is the service responsible for remote connections on servers (OpenSSH)

# Alias command

I can use the **Alias** command if I have a long command and I want to replace it with a small word.

**alias mycommand=’ifconfig eno1 | grep inet | grep -v inet6’**

But here, our alias is our temporary to our bash terminal so if we close the bash we will lose it. If we want to make our alias persistent, we must add it to my **bash profile** script so it would execute each time I log in.

# dnf

**dnf history info 7**

# Difference between yum and dnf

Dnf is installed by default on Fedora distribution and yum on rhel and centOs. In fact, dnf is the upgraded version of yum. We can use the same arguments with dnf and yum. In fact, rpm does not catch dependencies, so for example if I download an rpm it will have an error when we try to install it if we do not have the dependencies for the package.

Yum find its repo in the **/etc/yum.repos.d** . dnf find also its repo in this same repo of yum.

# Search packages with yum

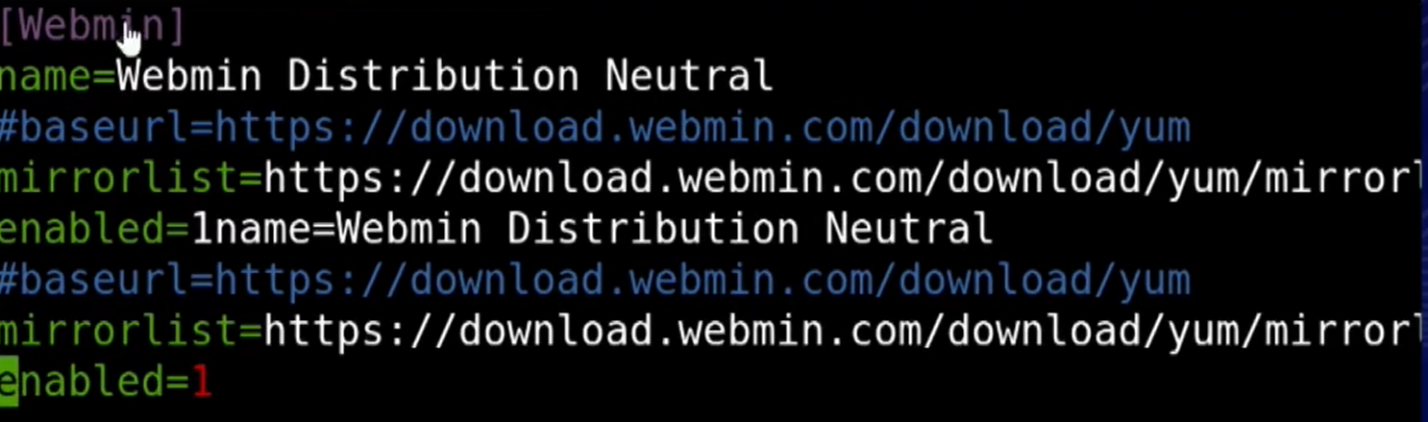
**yum search** **apache**

# Get more info about a package

**Yum info httpd**

# Add our package to the yum repo

In fact, in the package that we want to install is not in the yum repo, we should get it from the outside so from the vendor. Let’s say we installed the rpm package from the site and we want to install it. If we install it manually with the **yum install** …rpm we will not know about the updates of this package later on, that’s why w should add the package to the yum repo. To do that, we pass to the **/etc/yum/repos.d** and we create a new file and we can add the following to add the package webmin :



The name here means that yum and dnf both can use the name Webmin. Here we did not use the base url which tell us from where the repo are going to be taken and that is because we have the mirror list just after it so this is a list of servers that we can use to download the repo.

An important point to explain here is that how can we know that theses mirror list servers can be trusted to download the package? What we must do is to download the public key from the webmin website and import this key with the following commands :

**Wget** [**http://www.webmin.com/jcameron-key.asc**](http://www.webmin.com/jcameron-key.asc)

**Sudo rpm --import jcameron-key.asc**

And now when I do the install he will compare the key that I added to my list with the key inside of the repo itself

**Yum install webmin**

# Does yum automatically update the packages?

In fact, yum trace the packages but does not update them automatically, we can run cron jobs like cron yum but red hat does not recommend that, we should test the packages before put them on production.

**Sudo yum update** : This command will look all the repos and it is gonna compare the softwares that I installed with the softwares found in these repos.

With apt so with Ubuntu we had 3 commands to do that so we did apt upgrade then apt update and apt-dist upgrade to update the whole software. However, with yum they are all in one. So it is updating my cache, updating my packages and updating my OS.

# RPM and YUM

Understand RPM package :

bash-completion-2.7-5.el8.noarch

Package name

Version

Update

Patch nb

Linux version

Package type

♣ noarch means no Architecture so it works on 32 and 64 bits.

● RPM : Red hat package manager

Does not resolve dependencies

Manual updates

Difficult to manage installing/upgrading packages

● YUM : Red hat package manager

Does not resolve dependencies

# RPM

**rpm -qa** : pour rechercher les rpm des packages

# Building and installing from source code

So here we should the code to binary and then compile it.

I will need the 3 following packages : gcc, make , gzip

**gcc** is used to compile but it has a ton and a ton of parameters so that is why we use the command **make** that pulls a configuration and compile it. We might need also **gzip** because sometimes the code is compressed.

Normally, the source code in the src repository. We will see for example .c and .h files. The .c files are the actual C code that has been written and the .h are includes so it points to other libraries and other things.

So now I have all the source code in the src repo but I need to turn it to binary and compile it and that will be done with the utilities that I mentioned in the beginning of this section.

Normally, in this src repo , there is a file called Makefile.dep for example. And if this file is there, I can do the following command : **make config** . So this will do automatically the **gcc** command with all the parameters needed without us manually inserting that.

Here we might get an error saying something related to architecture. So here we can execute the command **make**. And this command will show us all the architectures to which we want the source code to be built to. We will choose the architecture of our machine : **make linux-x86-64**

After doing all the compile thing, normally it will create a repository called **run** just outside of the **src** repo.

After that I can access the **run** repo and execute the following command : **make install** . So all the things existing in the **run** repo will be sent to its correct places like /etc and other in our system.

# What command can usually be run, especially after a previous make command ran with an error, to erase the left-over files created while attempting to compile the software?

**Make clean**

# Increase security

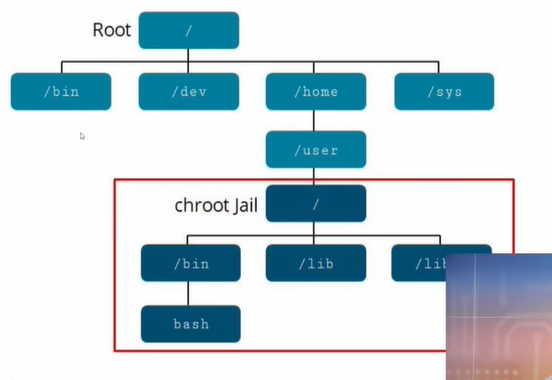
- Let’s say our web server was hacked so the hacker has access to the **/var/www/apache** . We can use something called **Chroot jail** which is shown in the photo below and what that will do is that for the hacker he won’t be able to see the real actual **/** repo but instead he will see the **chroot jail /** repo.

Another thing that the hacker may do is to cover his tracks so he will delete the logs. But me as a cybersecurity enginer I can have file auditing turned on that’s collecting everytime a file changes and then I can take that which Is all going to the syslog and we can use our syslog to send the logs to another system so in that way even if they flashed the logs on your web server we have a backup somewhere else, they can’t cover that also.

**Yum install audit** : Package used for file auditing.

So now I will be able to see in **/etc/audit/audit.rules** which repos are being monitored with th file auditing so for example if my **apache** repos are not being monitored I should add an entry in the **audit.rules** file.

The audit logs are stored in **/var/log/audit**



- Another way to secure our machine is by encrypting the hard disk and to do that Linux offers a tool called **luks** which is compatible with all linux kernels but this we do it when with the first installation of the linux software on a machine. To show an example I can create a new partition on the hard drive and do that.

I will use **blkid** command to see all the partitions then I will choose one and unmount it :

**umount /dev/sdb1** : So here I unmounted a partition.

**shred -v --iterations=1 /dev/sdb1** : So this command shred is used to really clear a hard drive and normally we make 3 iterations so it will replace values with 0s and 1s

**cryptsetup --verbose --verify-passphrase luksFormat /dev/sdb1**

So here the **--verify-passphrase** will do a multiple verification of the password, the **luksFormat** will format the disk to be a encrypted disk.

Now we can not directly copy files to that disk, we should mount it to unlock it and then put a file system in it.

**cryptsetup luksOpen /dev/sdb1 storage 1** : In fact, **storage1** is the virtual name like a mapper name to mount that virtual name.

**mkfs.xfs /dev/mapper/storage1** : Here we are creating a file system to that partition.

**mount /dev/mapper/storage1 /mnt/storage1** : So here I mounted a directory which is the **/mnt/storage1** to the partition which has a virtual name or mapper **/dev/mapper/**

**storage1**

# Blkid command

This command show us a list of all the disks I have.

# shred command

**shred -v --iterations=1 /dev/sdb1** : So this command shred is used to really clear a hard drive and normally we make 3 iterations so it will replace values with 0s and 1s

# SSH authorized keys

**.ssh/authorized\_keys** is a file that allows you to add ssh public keys of users that should be allowed to log into your server (the server in which the authorized\_keys file lives) using key based auth.

There is 2 methods to add the keys :

* Use the **ssh-copy-id**
* Add the key manually to the file **.ssh/authorized\_keys**

# SSH known hosts

**.ssh/known\_hosts** is a file that contains a list of keys from...known hosts that you have logged into. These keys pair an IP address with a server's key to help prevent you from logging into an impersonated server. Usually these keys will be setup the first time you log into a host but you can also manually configure these if security is very critical in your environment.

# Hardening SSH

In this section we will see how we can make SSH stronger. In fact, the ssh config files are in **/etc/ssh**. Here we will also see some public keys. However, we should be careful because in this repo we will find a config file named **ssh\_config** and another called **sshd\_config**. The first one is the client and the other is the server. By default, the port number for SSH is 22. If I want to change that, I should edit the **sshd\_config** file where I will see an entry that is commented and saying ‘Port 22’ so here I should change the number 22 if I want to change the port of SSH. But likely, it is the client that can choose which SSH version he wants to use so the server must be compatible with both versions of SSH. The value **Protocol** must be found but not obligatory in the **sshd\_config.** So we can have an entry like **Protocol 2** in that way the server will only be compatible with version 2.

Difference between SSH v1 and v2

In fact SSH v2 is more secured because it uses a much larger key than SSH v1.

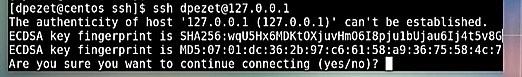
In the **/etc/ssh/** repo we will se public keys like **ssh\_host\_ecdsa\_key** and another the same but wit **.pub** extension so **ssh\_host\_ecdsa\_key.pub**. In fact, the first one is the private key and the other is the public key. By default, on our machine when the system was installed he create 3 different pair of keys so 3 differents strength (rsa pair key, ed25519 pair key and ecdsa pair key). But the question that comes here is that can we actually trust these keys? It depends on how the keys were generated in the beginning. There are keys that are generated randomly by using hardware so using the **/dev/random** but if the method of generation of keys happens on reboot for example the generated pair keys will be generated randomly by software and not hardware so here it uses **/dev/urandom** and here it is not so secured because computers are not so strong in generating random numbers so there may be exploits that can predicts these pair of keys.

So to fix all these problems and be sure that our pair keys are reliable we can generate ones. To generate keys we can run the following : **rm -f \* key\*** : So here we deleted the pair keys by default. Public keys are generated based on private keys. Now we will generate the pair keys :

**ssh-keygen -t rsa -f /etc/ssh/ssh\_host\_rsa\_key**: So the -t specifies the type, the -f specifies the file name of the key and where we want to put it. Here it will demand to enter a passphrase but we will keep it blank otherwise the sshd so the server will not be able to use it because he will not have the passphrase.

How users will be able to connect to the machine with SSH?

Each machine plays the role of SSH client and server. So if we want to connect to a destination machine, this destinated machine will be the SSH Server, so its public key must be accessible to the outside to the clients. So when we first connect to a SSH server we will see the following:



We can see the ECDSA key fingerprint and if we documented the Destined machine fingerprints of its keys I can verify if it is the same that is shown here so in that way I can know if it is really the machine that I want to connect to. But to place the fingerprints in a good place we can actually send it to all the clients so we will not be seeing this message again because the fingerprint of that server will be recognized on the client machine. In fact this file is in the home directory of our machine and it is hidden we can find it in the **.ssh** repo and here we will have a file named **known\_hosts** that will contain the trusted machines. I can add that public key manually to my **known\_hosts** file by using the following :

**ssh-keyscan** 172.16.119.15 >> **~/.ssh/known\_hosts**

In fact, when we say yes to the message above it will also insert the public key in th **known\_hosts** but can we really just trust these keys likes this? That’s why we can add manually these public keys when we are sure that these are the public keys so use the **ssh-keyscan** before really connecting and after that there is an option on the SSH client side that says that we can not connect to a machine without knowing the public key in advance so the key must be already in this file. So to do that we will edit the config file **vi /etc/ssh/ssh.config** and we have an entry saying **StrictHostKeyCheckng ask** so if we just remove ask we will force the client to have the public key of the destination server in the **known\_hosts.**

Okay so now that we have the public key locally on our client user I can copy it to all users of that same machine by using the **/etc/skel** which contains a template : Everytime a linux user is added it copies the home directory from this repo. So I can copy the **known\_hosts** in this **/etc/skel**

# su **vs** sudo

With **su**, we must enter the ‘root’ password so the administrator needs to give his root password to other users. However, with **sudo**, the user must be in the sudoers file so that he won’t need the root password of the admin.

**su -** elie

The - is used to execute all the scripts by default with the user switching.

# Parameters of sudo

**root ALL=(ALL) ALL**

**Username MachineName = (EffectiveUser) Command**

*User\_name: This is the name of ‘sudo‘ user.*

*Machine\_name: This is the host name, in which ‘sudo‘ command is valid. Useful when you have lots of host machines.*

*(Effective\_user): The ‘Effective user’ that are allowed to execute the commands. This column lets you allows users to execute System Commands.*

*Command: command or a set of commands which user may run.*

# Check ip address

**ip addr** : To see the interfaces and ip addresses.

# Relative path vs Absolute path

Absolute path is when we start from the root directory and pass to another directory (/home/elie)

# Difference between soft and hard link

Soft link is like a shortcut in Windows. In fact, in linux each file has a Inode. When we create a soft link we create also a inode to this file that is pointing to the the inode of the original file but if we delete the original file, the soft link will be useless.

Hard links are the same file with the same Inode than the original file but with different name, so if we delete the original file we will always have the hard link with its Inode so a copy of the original file.

# Soft link

* Unlike hardlink, with soft link we can see that it is actually a link : lrwxrwxrwx

How ever with hardlink we see -rwxrwxrwx.

* With softlink when we do **ls -l** we see also the arrow that points to the file.

Unlinke hard link, A soft link can cross a file system boundary and it can be used to link directories, as it simply uses the pathname of the destination object.

**ln -s** softlinksource softlinkdest : The ‘softlinkdest’ is the shortcut for the ‘softlinksource’ file

**stat** softlinkdest : We will see that size is equal to 0

**stat** softlinksource : We will see the actual size of the file to which we are pointing.

If we do changes on dest file we will have same content on source file and vice versa

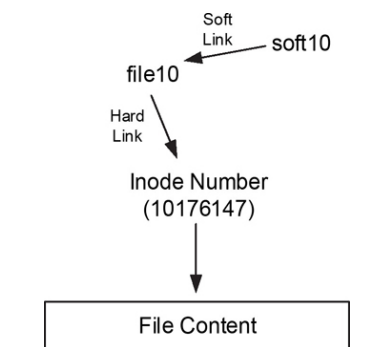
**rm -rf** softlinksource

**cat** softlinkdest : It will not work because the original file is no longer accessible. However, the link will stay but points to something that does not exist.

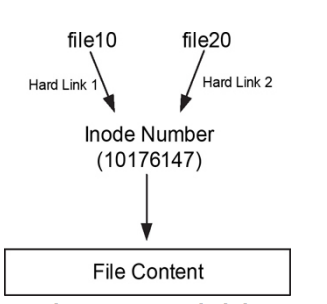
♣ Same thing with directories

**ln -s** Kenlm/LICENSE : Create link to that file with same name

-With softlink, the link created will have an inode number dedicated to him not like hardline where the hardlink will point to the same inode number as the original file.



# Hard link



Unlike soft links (Which generates a new inode number for each new link), each time we add a hard link, the counter of link of a file/directory that we see with the command **ls -l** will increase by 1. By default, the counter is 1, so if we add 2 links for that file/directory the counter will be equal to 3.

**ln** hardlink destlink : Now when I execute the command ‘stats’ on the destlink we will see the field ‘links’ having a value equal to 2 which means that it’s linking to the source (hardlink) and to the data. So when we delete the source, we will still have the data.

♣ Commands are case sensitive

♣ Hard link does not work on directories like soft link does

♣ Hard link does not have the typle ‘l’ that means link when we run the command ‘ll’ but soft link does show us that it is of type ‘l’. So Hard link is of type file because it has ‘-‘

When we create a hard link we will see the following:



We created a hard link to the original file which is ‘2014Sales’ after creating this hard link we see that we have a value of 2. If you remove the original file you will still have access to the data through the hard link created.

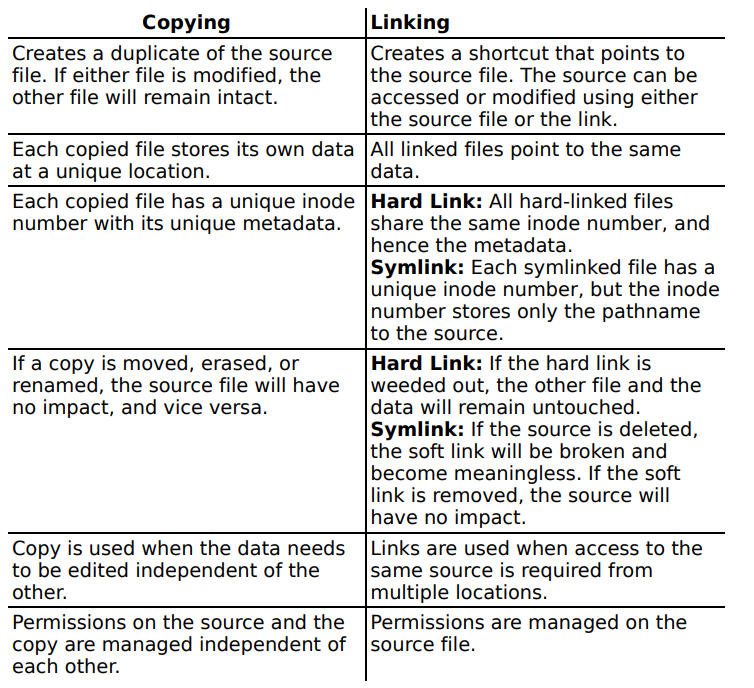
# There are two hard linked files in a directory. How would you identify them?

You can identify them by running ls -li.

# Soft linked directories cannot cross file system boundaries, but hard linked directories can. True or False?

False. Soft linked directories can and hard linked directories cannot cross file system boundaries.

# Difference between copying and linking



# Open a file and write in it

**cat >** firstFile

Here we can write some text but if there were some info in the file before it will be overridden.

**cat >>** firstFile

Here we will not override we will just add.

**cat** /etc/sudoers **>>** /opt/sudoers : Backing up to /opt/sudoers

**cat -n >>** firstFile

Will add the number for each line

# Time execution

**time ls -la** : Shows the time execution for ‘ls-la’

# ns lookup

**nslookup** google.com : Show us DNS server and other details (Resolves name to IP address)

**nslookup** 218.58.137.46 : Resolves IP address to name

In fact with nslookup we don’t have a lot of details. We can use the command :

**dig -x** google.com : Will show us mail server of google.com

# Tree command

**tree** /root/ : Will show us all subdirectories

**tree /**L1 : See the ‘/’ on level 1

# Traverse between directories

**cd -** : Go to the last directory.

**cd ~** : Go to home directory.

# Work with directories

**mkdir -p** 1/2/3/4 : Here we created 4 direcotires nested in each other so the the directory 1 is the first parent.

**mkdir** {2008..2021}-{01..12}

**rmdir 6/ :** Removes the directory 6 but in condition that the directory is empty otherwise we should add the parameter **-f**

**rmdir -rf** 4/ : ‘r’ means recursif so here it will delete nested directories. We can also add the **-i** that will ask for confirmation before deleting

# wc command

**wc** /etc/profile : Displays number of lines, words and characters.

-l Prints a count of lines

-w Prints a count of words

-c Prints a count of bytes

-m Prints a count of characters

# Check the history of commands

**history** : Show all executed commands and each command will be listed on a line with the number of the command, let’s say I want to execute the command of the line 71 I can do : **!71**

**!!** : Will execute the last command

**history -c** : Will delete the history

♣ The bash history file of the root user is saved in /root/.bash\_history We can also check while visualizing the environment variable : echo $HISTFILE

# Check info about machine and os

**uname -a**

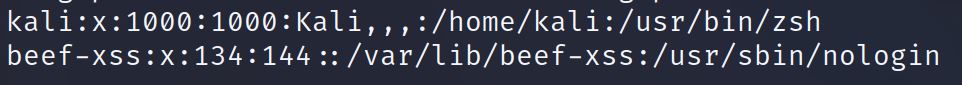
**cat** /etc/os-release

# File ownership

- File owner can only be changed by **root** user but access permissions can be changed by both **root** and file owner

- Shadow passwords improve system security by making passwords encrypted from

**/etc/passwd** to **/etc/shadow** and it is readable only by the **root** user.



So this file will show us the following : **user Name**, **password** **user ID**, **group ID**, **Description,** **Home directory** and **shell**. The group ID is stored in the **/etc/group**

In fact the root user has an ID of 0 for the user ID and group ID.

Owner : Group : World : Path

4 read (r)

2 write (w)

1 execute (x)

7 = 4+2+1

6 = 4+2

**ls -l** file ou **ll** file:

- rw- rw- r--

The - can be either ‘d’ which stands for directory or ‘l’ that stands for link or ‘-‘ that means file.

**chmod** 554 file : Will give the following access rules for the file.

**chmod +x** : Add execute permission to owner and group of file.

**chmod u+rw** : Add r/w to user.

**chmod -R+x** : Add execute permissions to files and directories permissions recursively.

**chmod g-rw** : Removes read and write for group

**chmod ugo+rwx** : Adds r/w/x for user,group and others.

**chmod -R 770** Downloads/ : Changes recursively the permissions for this directory.

**chown** root file : Change the owner group of file so root is owner of file.

**chown** root : root file : Changes the owner and group of file to root.

**chgrp** root file : Change the group to root for the file.

# touch command

**touch** file-‘date +%d-%M-%Y’

Result : file-4-12-2021

**touch -d** 2019-09-20 file1 : Sets time to file

**touch** file{1..100}.txt

Here we will create files going from 1 to 100

# Copy some files

**cp** .txt Downloads

Will copy all text files to the Downloads directory.

**cp -pv** file Downloads/

The ‘-p’ is used to preserve parameters so for example the modification date of the copied file will be same as before copying. The ‘-v’ means verbose which will show us a sentence explaining what happened.

**cp -r** Desktop/ Downloads/

The ‘-r’ is used to copy a directory.

By default, the copy operation overwrites the destination file if it exists without presenting a warning. To alter this behavior, use the -i (interactive) option to instruct cp to prompt for confirmation before overwriting:

**scp** backup.tar.gz root@192.168.2.140:/opt/ : Will copy file to opt directory on the remote server.

# Move some files

**mv** file othernamefile : Renames the file.

**mv** file Downloads/ : Moves to Downloads.

# Remove some files

The rm command can also be used to remove a file that has a wildcard character, such as an asterisk (\*) or a question mark (?), embedded in its name. These characters have special meaning to the shell, and filenames containing them must be prepended with the backslash character (\) to instruct the shell to treat them as regular characters

For example, if a file exists by the name \* under the /tmp directory (use touch /tmp/\\* to create it), you can remove it by executing rm /tmp/\\*. If you mistakenly run rm /tmp/\* instead, all files under /tmp will be deleted.

# List some files

**ls -li** /tmp/hard1 : ‘-i’ is used to display inode number.

**ls -m** /home/elie : ‘-m’ is used to list files separated with a comma. Usually used when exporting.

**ls -R** : list subdirectories also.

**less** file.txt : Display file content 1 page at a time and we can jump to other page with space button.

**more** file.txt : Do the same thing but here we can see the percentage so when we jump to another page we can see how much pages left we got to see

# Remove some directories

The **rmdir** and **rm** commands remove directories. The **rmdir** command is used to delete empty directories, while **rm** requires the **-d** option to accomplish the same. In addition, the **-r** or **-R** (recursive) flag with rm will remove a directory and all of its contents

# Difference between less and more

More has some limitations, in fact with more we can not scroll back so once we jump to another page we can not go back, which is not the case with thes less command. But more give us the functionality of percentage.

Spacebar / f Scrolls forward one screen

Enter Scrolls forward one line

b Scrolls backward one screen

d Scrolls forward half a screen

h Displays help

q Quits and returns to the command prompt

/string Searches forward for a string

?string Searches backward for a string; only applies to the

less command

n Finds the next occurrence of a string

N Finds the previous occurrence of a string; only

applies to the less command

# Compare two files

**diff** file1 file2 : Verify if two files are equal

# Output some data

**head** : By default, prints first 10 lines of file

**head -n** 15 : First 15 lines

**tail** : last 10 lines

**tail -n** 20 /var/log/messages **>** /home/aravi/logfile

**tail -f** /var/log/messages : Watching a log file while it is being updated in real time.

# Arithmetic expression

**echo $**((7+3)) : Will output 10

# Brace expansion

**echo** abc{elie,joe}xyz : Will output 2 things : abceliexyz and abcjoexyz

**echo** {090..100} : 090 091 092 … 100

**echo** a{A{1,2},B{3,4}}b : aA1b aA2b aB3b aB4b

# Work with echo

**echo $**{USER} : Will output the value of the variable USER

**echo \**$USER : Outputs \USER

**echo** my name is \\$USER : Will output my name is elie because the \\ will cancel the one the other.

# Work with partitions

**df -h** : Will show us partitions and directories with some details.

**du -sh** /etc/\* : Shows directories size inside of etc directory in human readable format.

**df -x** tmpfs : List the actual file system details, not the temporary file systems.

**df -T -x** tmpfs : To see type of file systems also

# Work with users

**who** : See the users logged to the machine.

**whoami** : Show the actual user that is logged.

**w** : Similar but gives more details like when user was logged in and out…

# Work with processes

**ps -aux** : Show all running processes

**ps -U** elie : Show the processes running by the user elie.

# top command

**top** : This command will output the same as ps aux but it in a more clear way and it will sort for us the utilization of each process by order

# htop command

**htop**: Comme **top** mais ca nous permet d’avoir une vue graphique des process.

# Uptime command

**uptime** : See when machine (server) was rebooted and how many user are logged in…

# tac command

The **tac** command is like **cat** but reversed so it will output content from the bottom to the top.

# Get help for a certain command

**ls --help**

**man ls**

# Search for files

**locate** or **find** : Locate is faster but need database update. Find is used when we know where the file is located.

**updatedb** : Will do a database update

**find** /home/ **-i -name** “file1” : The ‘-i’ will ignore the case sensitive.

**locate -i** fil : If you know a fragment of file

**locate -ib** fil : Without the ‘-b’ it will take the path as a search parameter so it will output everything in it. The ‘-b’ will force to use the last thing we wrotes in the command.

# Unarchieve and unzip a file

**tar -xvzf john-1.8.0.tar.gz** : So here, it will unarchieve the tar and then unzip

# The tar command can be used to archive files with their SELinux contexts. True or False?

True. The tar command has the --selinux switch that provides this support.

# Archieve and compress

Archieving is different than compressing. Archieving means putting files into one file without reducing size so without compressing.

Quand on fait le tar par exemple :

tar -zcvf passbolt\_backup\_new.tar.gz /root/passbolt\_backup\_new/

le tar va prendre même le chemin donc quand on va le unzipper, il va créer les répertoires root et passbolt\_backup\_new donc il copie le path aussi. Si on veut juste qu’il nous extrait les fichier sans créer des repos il faut se situer dans le bon endroit et faire :

tar -zcvf passbolt\_backup\_new.tar.gz .

We have 3 types of zip methods :

- gzip

- bzip

- zip

We can archeive+zip in one method :

- tar

**tar -cvf** archieve.tar varlog.tar.gz anothervar.rpm : The “-c” means create, the “-v” means archeive and the “-f” means specify files. So here we archieve files into one file without compressing.

**tar -cvzf** archieve.tar varlog.tar.gz anothervar.rpm : The “-z” will zip the resulting file.

**tar -uvf** archieve.tar \*.txt : We can update the contents of the archieve file so for example if we made changes in the folder and want to recompress it we can just update. Here we are saying to check all text files that has been changed to update the zip folder in the end.

**tar -tf** archieve.tar : The “-t” will list the files that are present in the archieve folder.

**tar -xvf**  archeive.tar : Extracts files from archieve.

**zip** -9 **-r** test1.zip /home/aravi : Here we used zip so it is another method and “-9” let us do high compression and here we are compressing all subfiles into one archieve which is tes1.zip

**zip -d** test1.zip home/aravi/ioen/testing.txt : The “-d” is used to delete specific file from the archieve.

**zip -fr** test1.zip \* : Here the test1.zip will be updated.

**unzip** test1.zip : To unzip file

# Redirection of errors

Standard input : 0

Standard output : 1

Standard error : 2

**ls** /root 2**>** /temp/errors : Will insert the error line into errors file and the ‘2’ refers to the standard error value.

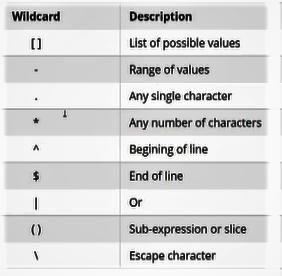
**find** /etc **-name** passwd **>** /tmp/output 2 **>** /tmp/error : Will redirect the output file to /tmp/output and if there is an error, the /tmp/error will be filled with the error line.

**ls** /home/aravi **>>** /tmp/errors 2 **>** &1 : The **>>** will let insert multiple things instantly so here we insert the output and the error if there is one.

# Pass output as input for a command

**grep** aravi **<** /tmp/output : Will give the output of /tmp/output to the command as an input

# Grep command



The last symbol that we can see here which is the ‘\’ is used to escape character. For example, if I want to search for a dollar sign contained in a file so lets say it’s the currency dollar that we are looking for, we can directly use the $ sign because it is used as a symbol by the regular expression in that case we must use the escape character to fix the problem.

By default, grep uses fgrep which does not use extended regular expression. If we can to us extended regular expression we must use the -**E**

**cat** cal-2019.txt **| grep -E** “Halloween|Christmas” : So here we used regular expression so we used the -E to say extended regular expression we could also use directly the command **egrep** and we used also the “­|” which means OR. So here we are searching for any line containing The word Halloween or the word Christmas.

**cat** cal-2019.txt **| grep -E** “^1[0-2]” : So here we are searching for any line that starts with 1 and then may have the second character which can be either 0, 1 or 2.

**grep -e** “First” **-e** “last” demofile : So the ‘-e’ must be used if we want to search multiple strings

**grep -v** “First” demofile : The ‘-v’ is used to unmatch the word so we will output everything except the word “First”.

**grep** “First” **-A** 2 file : Will output line containing the word “First” and 2 lines after.

**grep** “First” **-B** 2 file : Will output line containing the word “First” and 2 lines before.

**grep** “All” **-C** 1 file : Will output 1 line before and 1 line after.

**grep -ril** /home/aravi/ : The “-l” is used when searching files or directories. The ‘-r’ stands for recursvie and the ‘i’ means ignore case sensitivity.

**grep -l** demo\* : Will output the files having “demo” included in their name.

**grep -c** first demofile : Will output the number of times the word “first” is found in demofile.

**grep** “demo**$**” demofie : Will output the line that ends with the word “demo”.

**cat** /var/log/messages **| grep** “^Sep 26 03:26:41” : Will search for lines beginning (^) with date.

# Vi&Vim editor

Vi is the best editor to use in Linux. It is used to modify config files.

The difference between Vi and Vim is that Vim is an imoproved version of Vi where output is colored.

**3 modes :** command mode (Mode par default like ESC-key mode), input mode, last line mode(When u enter : while being in the commande mode)

i Inserts text before the current cursor position

I Inserts text at the beginning of the current line

a Appends text after the current cursor position

A Appends text to the end of the current line

o Opens a new line below the current line

O Opens a new line above the current line

h Moves backward one character

j Moves downward one line

k Moves upward one line

l Moves forward one character

w Moves to the start of the next word

b Moves backward to the start of the preceding word

e Moves to the ending character of the next word

$ Moves to the end of the current line

Enter Moves to the beginning of the next line

Ctrl+f Scrolls down to the next page

Ctrl+b Scrolls up to the previous page

0 (zero) to move to the beginning of the current line

[[ to move to the first line of the file

]] to move to the last line of the file.

**Deleting Text :**

x Deletes the character at the cursor position

X Deletes the character before the cursor location

dw Deletes the word or part of the word to the right of

the cursor location

dd Deletes the current line

D Deletes at the cursor position to the end of the

current line

:6,12d Deletes lines 6 through 12

You can precede any of the commands listed in Table 3-5, except for the last line mode command, by a numeral to repeat the command action that many times. For instance, 2X would delete two characters before the cursor position, and 3dd would delete the current line and the two lines below it.

**Undoing and Repeating :**

u Undoes the previous command.

U Undoes all the changes done on the current line.

:u Undoes the previous last line mode command.

. (dot) Repeats the last command run.

**Searching for Text :**

/string Searches forward for a string

?string Searches backward for a string

n Finds the next occurrence of a string

N Finds the previous occurrence of a string

**Replacing Text :**

:%s/old/new Replaces the first occurrence of old with new in a

file. For example, to replace the first occurrence of

profile with Profile, use :%s/profile/Profile.

:%s/old/new/g Replaces all occurrences of old with new in a file.

For example, to replace all the occurrences of

profile with Profile in a file, use

:%s/profile/Profile/g.

**Copying, Moving, and Pasting Text :**

vim allows you to copy some text and paste it to the desired location within the file. You can copy (yank) a single character, a single word, or an entire line, and then paste it wherever you need it

yl Yanks the current letter into buffer

yw Yanks the current word into buffer

yy Yanks the current line into buffer

p Pastes yanked data below the current line

P Pastes yanked data above the current line

:1,3co6 Copies lines 1 through 3 and pastes them after line 6

:4,6m9 Moves lines 4 through 6 after line 9

You can precede any of the commands listed in Table 3-9, except for the last line mode commands, by a numeral to repeat the command action that many times. For instance, 2yw would yank two words, 2yy would yank two lines, and 2p would paste two times.

**Saving and Quitting vim :**

:w Writes changes into the file without quitting vim

:w file2 Writes changes into a new file called file2 without

quitting vim

:w! Writes changes to the file even if the file owner

does not have write permission on the file

:wq Writes changes to the file and quits vim

:wq! Writes changes to the file and quits vim even if the

file owner does not have write permission on the

file

:q Quits vim if no modifications were made

:q! Quits vim if modifications were made, but we do

not wish to save them

# Nano editor

Vi is more complicated than nano with its key bindings. Nano is like windows notepad.

- Provides more features than pico

- Colored text for writing scripts

- Smooth in scrolling

- Simple control keys

- Regular expression support to search text in file

**nano** file1

# Manual page

We can use the **/toto** and then with the **‘n’** key we can pass from the found value to the other and with the **‘ctrl+n’** we can go back to previous value.

**man -s** 5 passwd : Will give us passwd(5) which is the 5th sub category.

**man -k** printf : Shows us how many related command are there. So all commands related to printing.

**man -s** 5 **-k** passwd : Show the command linked to passwd in the 5th sub category.

♣ In the manual page I can enter / and search.

**pinfo** ls : Similar to man command but different the way and content.

**info** ls : Exactly similar to man command.

# whereis and whatis command

**whereis** python : Show us where it is located

**whatis** ls : Brief description of command

# SSH Keys

Here we

# SSH Servers

Here we are going to explore OpenSSH server&client configuration so how to securely log in to the server and how the client is authentified. OpenSSH use port 22 by default for communicating between client and server. Public and private keys are used for cryptographics purposes.

- Firstly we must install the OpenSSH package :

**yum install** openssh

♣ Difference between yum and rpm : Yum is a package manager and rpms are the actual packages. With yum you can add or remove software. The software itself comes within a rpm.

**rpm -qa | grep** openssh : Will show us the installed version of openssh, so here if we are on the server side we will see the installed package.

- Now we have to configure the config files and SSH keys on server side and authenticate the client.

**systemctl status** sshd : Will show us the status of the service openssh.

**systemctl enable** sshd : To enable the service openssh if it is not enabled.

- Add the service to the firewall :

**firewall-cmd --permanent --add-service=ssh**

- Now we must verify the config files :

**vi** /etc/ssh/sshd-config : Now when we do access of the file we can add some code

a) Add the protocol type responsible for a secured environnement :

Protocol 2

UsePAM yes

AllowGroups sshusers (This group only will have access to the ssh server)

- Now we will create the group

**groupadd** sshusers

**usermod -Ag** sshusers aravi : Add user aravi to group sshsuers

**id** aravi : To verify that aravi is part of the group sshusers.

- Now we will try to connect to the server

**ssh** aravi@192.168.2.140 : Here he will ask for passwd of aravi and we will be able to connect to ssh server.

♣ I can also use Putty and choose the connection type as SSH and he will also ask for user’s password.

# SCP (Secured copy)

In this section, we will learn how to securely copy files from a source to another.

- Source IP Server : 192.168.2.140

- Destination IP Server : 192.168.2.42

**scp** file1 root@192.168.2.42:/root/ : The ip address and location in orange is the ip address of the destination server and the location where we want to store the file.

We can add multiple files simulatenously

**scp -r** directoryName root@192.168.2.42:/root/ : Here we used the “-r” to copy entire directory.

**scp -C** filename root@192.168.2.42:/root/ : The “-C” is used to compress and send data.

In fact, in this situation the timestamp will be the actual one when we make the copy that’s why we can use the parameter “-p” to preserve parameters.

**scp -vp** filenameroot@192.168.2.42:/root/ : To preserve parameters

**scp -l** 500 filename root@192.168.2.42:/root/ : To limit the bandwidth allocated for the copy.

# Linux Processes

**sleep** 300 **&**

output : [1] 2798 : This is the process ID.

**ps -aux | grep** 2798

In fact, each process has a specific state :

A) Running : Running or ready to run

B) Waiting : Process is waiting for an event or for a resource (Like waiting to be allocated a certain part of CPU or Ram.

C) Stopped : Received a stop signal

D) Orphaned : Process exists while children still running, thoses childrens are orphanes.

E) Zombie : It is a halted process. Still has a task\_struct data structure in task vector. .Here the process is dead but still found in Process table. To find these processes:

**ps -aux | grep** Z

**pstree** : Shows us the process tree containing father and children processes.

**ps -ef** : Show us not only processes like **ps -a** but also sleeping processes and all processes.

**ps -aux --sort = -pcpu,+pmem**: To sort process with utilization rate.

**ps -e -o** : Filter with , so we will see pid,uname,pcpu,pmem…

**jobs** : [1] Running

**fg %**1 : Here the ‘fg’ means foreground so it will kick the process having id ‘1’.

**kill -l** : Will show us all parameters that I can give to the kill command and each parameter is identified with a specific ID or number.

**kill** -9 3298 : Here we will kill this process. (-9) means kill process.

**pgrep -u** root ssh: See all executed commands with root processes about ssh.

Change process priroty : A normal user can only decrease a process priority but not increase it. The ‘-20’ is the highest priority value and the ‘20’ is the lowest.

**nice** -10 **sleep** 500 **&** : Will give a value of ‘-10’ as priority for this process.

# Hardware materials

**dmidecode -t 17**: Voir des info concernant le hardware qui est la RAM dans ce cas

# Check logs

**dmesg** ou **more** **/var/logs/messages**

# Check service errors

**journalctl -u nfs** (nfs est le nom de service)

# fdisk vs df -h

**df -h** shows us only mounted partitions however with **fdisk -l** we can see all partitions that exists on our drive (mounted or not)

# Disk partitions and file systems (CompTIA)

The are 2 types of partitions : The old style disk and the new style disk. The old one use something called MBR (Master boot record) which is implemented in the middle of the harddrive and that tells where the data is stored on the real harddrive so when the system boots up he will try to find the MBR and then search for the OS. With MBR, we are limited to only 4 partitions. Usually we put the home directory in Linux in a separate Partition and the system files partition in another, in that way if a user put a lof of things in the home directory it won’t have effect on the system partition. And also for the swap partition, we need a specific partition for that purpose. To fix that problem with MBR, they created something called Extended partition which let us create fake logical partitions inside of the a real partition.

Now the new style disk is GPT and with GPT we can create 128 partitions so we do not need to use the extended logical partitions like MBR does.

**lsblk** : To see all partitions, even USBs…

# MBR (CompTIA)

**sudo fdisk -l /dev/sdb** : List info about the disk

**sudo fdisk /dev/sdb** : Access the config

# GPT (CompTIA)

**sudo gdisk /dev/sdc**

# See all file systems supported to use (CompTIA)

**ls /usr/sbin/mkfs\***

# Create a file system on a partition (CompTIA)

**sudo mkfs.ext4 /dev/sdb1** : So here we used the **ext4** filesystem for our **sdb1** partition

# Create a swap partiton system on a partition (CompTIA)

**sudo mkswap /dev/sdb3** : So here the swap has a specific file system not like the **xfs** or **ext4** like the ordinary filesystem because the swap will contain RAM data.

**sudo swapon /dev/sdb3** : Activate swap

♣ Changing the file system of a partition will erase all present data so make sure to backup your data before doing that. However, there are some exception, for example if you are on ext3 you can convert to ext4 without formatting so it is just an extension that is being added. To do that change we can use the **ls /usr/sbin/tune\***

# Creating labels for partitions (CompTIA)

After creating the filesystem, we can create labels for our partitions to simplify the name of the partition. For example, we will change the name of the following partition to ‘public’

**sudo e2label /dev/sdb1 Public**

# exportfs

**sudo exportfs -a** : Nettoyer ce qui etait mounté

# Moutning with nfs

**mount -t nfs 172.16.118.128:/isos/RHEL /iso**

Here we created a directory called ‘iso’ on our local machine and we mounted from nwadmin the repository /RHEL.

# Moutning partitions temporarly (CompTIA)

We can not actually use or access a partition without mounting already that partition.

**mount** : This command will show us all mounted partitions on our system.

For example, the **/home** is mounted on a different partition than the **/** directory. So each directory in our system is mounted on a partition.

Actually, there are medias that are mounted automatically. For example, when we put a CD drive, it will be mounted automatically. All mounted things are present in the **/mnt** directory. We will create a directory called ‘Public’ in the **/mnt** directory and we will mount it to **/dev/sdb1**.

**sudo mount /dev/sdb1 /mnt/Public**

♣ What we have done here is temporary so it will be removed after reboot.

# Moutning partitions in a persistent form (CompTIA)

We should add entries in the **/etc/fstab** to tell the system to mount some things when he first launches the system.

**sudo vi /etc/fstab**

We will be mounting with 2 ways : One way is by using device names like **/dev/sdb** and the other way is to use **elabels** like an alias. But, sometimes the device name changes if we make hardware changes so the mount will be not be done successfully when system starts.

Method with device name (Entry to add in **/etc/fstab**):

**/dev/sdb1 /mnt/Public ext4 defaults 0 0**

Method with elabels (Entry to add in **/etc/fstab**):

**label=Private /mnt/Private xfs defaults 0 0**

# Disk partitioning

**sudo fdisk -l** : To see the hardrives or partitions.

♣ You can make only 4 partitions per hardrive (for Windows or Linux). In fact, we use the 4th partition for example as an extended one where inside of it we can use multiple partitions (locial partitions)

**sudo -s** : Login with root and go to home directory of present user.

**fdisk** /dev/sdb : Then enter ‘m’ for help then we will see multiple parameters to choose from and we will type ‘n’ to add a new partition. Then we will choose ‘p’ for primary then we will choose a partition number (1🡪4). Then we will choose the size of the 1st partition, we will put +5G then we will enter ‘wq’ to write and quit.

**partprobe** /dev/sdb : To update the partition table to this hardrive.

Now we will create or directory and add it (mount it) to a specific partition.

**mkdir** /part1

**mount** /dev/sdb1 /part1/ : It will not work and return an error, we need to add a file system to the partition.

**mkfs**.ext4 /dev/sdb1

**mount** /dev/sdb1 /part1/

♣ Difference between ‘df -h’ and ‘fdisk’ :

df -h : Displays all the mounted partitions.

fdisk -l : Displays all partitions that exist on your disk. Apparently, there are some partitions which exist but which are not mounted

**df -h** : To see all mounted partitions. Now here we will notice that the mount didn’t actually work, we should add something in /etc/fstab

**vi** /etc/fstab

Then we will add the following line :

/dev/sdb1 /part1 ext4 defaults 0 0

**mount -a**

**df -h** : Now we will see that part1 is mounted to sdb1

# Delete partitions

**vi** /etc/fstab

Then we will delete the following line :

/dev/sdb1 /part1 ext4 defaults 0 0

**unmount** /part1

**fdisk** /dev/sdb : Then ‘p’ to print partitions then ‘d’ to delete and choose the partition number, we will choose ‘1’ which refers to sdb1 then ‘wq’ to save and quit.

**partprobe** /dev/sdb

# Create LVM

We used standard partitions in the previous section. One of the disadvantages is that if we want to increase a partition size we need to stop the users using the partitions so the partition will be shut down until we increase it and finish and remount it again. LVM or Logical Volume Manager combines multiple hardrives so if we want to write a file instead of writing in on a single partition which will take specific time, with LVM the time will be split on the number of hardrives so the file will be written in equal manner on the hardrives and this volume group that combines multiple hardrives can be splitted on as many slices that we want. Each slice is called logical volume. So we will have logicalvolume1, logicalvolume2, etc..

In our example, we create 2 partitions. One of them is sdb1 and the other is sdc1. Sdb1 is the partition in sdb hdd and sdc1 is the partition in sdc hdd. In fact, these partitions are standard partitions. We need to convert them to LVM partitions. To do so :

**fdisk** /dev/sdb : Then ‘m’ then ‘t’ then ‘L’ to list all codes then ‘8e’ which is the code of Linux LVM. Now we converted the partition from standard to Linux LVM. Now we will pass to sdc.

**fdisk** /dev/sdc : Then ‘m’ then ‘t’ then ‘L’ to list all codes then ‘8e’ which is the code of Linux LVM. Now we converted the partition from standard to Linux LVM. Now we will pass to sdc.

**partprobe** /dev/sdb

**partprobe** /dev/sdc

Now we have to create the physical volumes inside of each partition

**pvcreate** /dev/sdb1

**pvcreate** /dev/sdc1

**pvs** : To see the created pv’s.

Now we will create the volume group

**vgcreate** VG0/dev/sdb1 /dev/sdc1

**vgs** : To see created vg’s.

**vgdisplay** VG0 : Will output further details on VG0.

**lvcreate -n** lv0 **-L** 4G VG0 : Here we created the logical volume containing the volume group. Now we will create another logical volume.

**lvcreate -n** lv1 **-L** 2G VG0

Now as we created pv’s then vg then lv’s we will create file systems and mount some things to them.

**mkfs**.ext4 /dev/VG0/lv0

**mkfs**.xfs /dev/VG0/lv1

**mkdir** /ext4part

**mkdir** /xfspart

**vi** /etc/fstab

We need to add the following lines :

/dev/VG0/lv0 /ext4part ext4 defaults 1 2

/dev/VG0/lv1 /xfspart xfs defaults 1 2

**mount -a**

**df -h**: To check

# Extend LVM

Here we will show how to extend filesystem without interrupting applications.

FREE SPACE

**LV0**  **Extend LV0** **VG0**

In fact, to be able to extend without interrupting user’s applications we need to resize the file system to this extended value.

**lvextend -L** +1G /dev/VG0/lv0

**resize2fs** /dev/VG0/lv0

♣ For xfs, the below command will change

**lvextend -L** +1G /dev/VG0/lv1

**xfs\_growfs** /dev/VG0/lv1

# Access Linux

● What is swap file system and how it works?

In fact, swap file is used to optimize the load on the physical memory. if we have some apps that are opened we may fill our memory capacities. In order to optimize that we will remove the apps that are inactive for a certain time and put them on the hardisk so we will always have the info but in another location.

Swap memory = Ram X2

Now we will create the partition as swap.

**fdisk** /dev/sdc : Then ‘m’ then ‘n’ then ‘p’ then ‘+2G’ then ‘t’ then ‘1’ then ’82’ which will convert the file system of sdc1 partition from Linux to Linux swap/Solaris. Then ‘wq’ to save.

**partprobe** /dev/sdc

**mkswap** /dev/sdc1

**vi** /etc/fstab

Then add the following :

/dev/sdc1 swap swap defaults 0 0

**swapon -a**

**swapoff -a** : To turn off swap

**swapon -s** : To see the swap partition.

# Create users

There are 3 types of users :

* Super User (Root/Administrator)
* SystemUser : UID : 1 999 (Appear when downloading a package service for example)
* Normal Users : UID : 1000 65 000

Useradd command will create by default multiple properties like Home directory, login shell details, by default there is also a group that is created with the same name of the user and there are some files that are created also like .bash…

And each time we add a user, we will have some files that will be modified like /etc/passwd and /etc/shadow (Contain the password that we added for this user but here it will be encrypted, and /etc/gshadow file will be modified if the group for which we added a user has a password allocated to it (By default, the file is not modified). /etc/skel

♣ A user can only have 1 primary group but can have multiple secondary groups.

**sudo -s** : To pass to the administrator

**cat** /etc/passwd : We will some entries that contains all users (We will see system users that are created for each and every service)

Now we will explain in details each field of a user (normal user) entry.

**cat** /etc/passwd **| grep** aravi

Output : aravi : x : 1000 : 1000 : Ravi Kumor :/home/aravi :/bin/bash

The field ‘aravi’ is the login name user used to log to the system. The ‘x’ means that password is encrypted and added to /etc/shadow file. The first ‘1000’ field is the UID (User Id). The other ‘1000’ field is the Primary Group Id. After that if there is some secondary groups, they will be listed here. We see ‘Rami Kumor’ after that which is the description of the user. ‘/home/aravi’ is the home directory for this user so when a user logs on he will be transferred directly to this directory. The last field is the shell used which is bash in this scenario.

There are some default config that has been allocated to usedadd command and we can see it with the following command :

**cat** /etc/default/useradd

We have also a file containing user group details and login expiry password age etc… in the following path :

**cat** /etc/login.defs

When we create a new user, by default a group with the same name will be created in /etc/group. The user is created in the home directory. By default, there will be some files in the user directory that are copied from /etc/skel directory.

Now, we create a user with some customization :

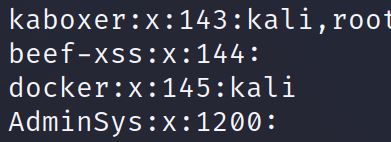
**useradd -u** 2000 **-g** 2000 **-G** aravi,test2 **-c** “Some user” **-s** /bin/bash **-d** /opt/rhesa **-e** 2018-08-28

**-p** elie@123 rhesa

‘-u’ defines UID. ‘-g’ defines primary group id. ‘-G’ defines secondary groups. ‘-c’ is for comments. ‘-s’ is for shell. ‘-d’ designes the home path for user. ‘-e’ is the user expiry date, after that he will be deleted. ‘-p’ is the password chosen and the last field is the name of user.

# Understand group file

The /etc/group file is divided with the following values :



For example, we see in the first entry the name of the group which is ‘kaboxer’ then the second value is the password. The third value is the Group ID and the fourth value is normally for the users that are existant in a supplementary group so for example if I create a new user and append him to a primary group this user will not be shown in this group in /etc/group

# Modify user properties

Here we used usermod to modify

**usermod -l** ravi test : Will change the login name from test to ravi.

Now we will add the user to same groups

**usermod -aG** aravi,root ravi : ‘-a’ means append so we are appending those groups to ‘ravi’.

Now we will change the primary group of ‘ravi’ user :

**usermod -g** aravi ravi

Now we will log the user, in that way he won’t need to enter the password when he tries to log.

**usermod -L** ravi : Locks a user

**usermod -U** ravi : Unlocks a user

In fact, when a user is logged, we can see in /etc/shadow a ‘!’ that will appear before the encrypted password.

**usermod -f** ravi : Sets the user inactive after password expiration :

To change a user’s password from another user, only the super user can do that so pass to root :

**passwd** aravi

**userdel** aravi : To delete a user (Here home directory will not be deleted)

To delete all user’s data :

**userdel -r** aravi

To delete a user from a group :

**gpasswd -d** aravi finance

To delete a group :

**groupdel** finance

To change group name :

**groupmod -n** newname oldname

Set expiry date for a user :

**usermod -e** 2018-08-22 ravi

To see password parameters, use the following command :

**chage -l** ravi

Here we will see the password expiry date

**chage -m** 0 **-M** 90 **-W** 10 **-E** 2018-09-28 ravi

The ‘-m’ is the minimum number of days between password change. The ‘-M’ is the maximum number of days. The ‘-W’ provide warning before 10 days of password expiration. The ‘-E’ is the expiry date for the password.

# Modify group properties

In this scenario, we will create a group, 3 users and we will add the users to the group.

**groupadd** finusers

**useradd** user1

**useradd** user2

**useradd** user3

**usermod -G** finusers user1

**usermod -G** finusers user2

**usermod -G** finusers user3

We can protect our group using a password, in that case if another administrator wants to modify our group he must know the password.

**groupmod -p** testfinusers

Now we will try to insert a new user in that group by another administrator. In fact, it will return permission denied and to fix that we will make the other administrator as the owner of the group.

**gpasswd -A** user4 finusers : The ‘-A’ means administrators.

We can add multiple users as admins at same time.

**gpasswd -M** user1,user2,user3 finusers

To remove a specific user as being the admin of group :

**gpasswd -d** user4 finusers

To check id and to what groups a user belongs to :

**id** user1

To modify group properties :

**groupmod -g** 1015 finusers : Will change group Id to 1015

**groupmod -n** engusers finusers : Change name of the group

To delete a user can go to /home/directory and execute the command :

**userdel** user1

But here we will still see the folder of user in /home/ because with this command, data will not be erased.

To delete all user data :

**userdel -r -f** user1

# Permissions

The defaullt permissions are :

File : 644

Directory : 755

**ls -ltr**

Here we will see the permissions for directories or files. After that we have a field which is a count and show us how many files are there in a directory, then we will see the owner of the user and the group which have permissions and then we have the size field, creation date and name of directory.

Output : d rwx r-x r-x 2 aravi aravi 6 Jun 2 16:30 Videos

If the first field ‘d’ was ‘-‘ it means that it’s a file. Here it is a directory and then we see the permissions for the user, the group and others successively.

♣ If you do not have a read permission on a directory then you can not go inside it.

When using the numerical value with chmod, always enter the 3 digits. If you do chmod 7 file for example, the permissions of user and group will be removed and others will take the value 7 which refers to rwx so output will be --- --- rwx.

In fact, when we create a file for example with the touch command, the file will get some permissions by default. Actually, these permissions are received because of the umask command.

**umask**

Output : 0002

The default value of a file is 666 and the default value of a directory is 777. We will now explain how and why the umask value is 0002. The output of ls -ltr for the file that we created with touch by default gets the following permissions : -rw-rw-r-- which is equivalent to 664. And we said that the default value of a file is 666 s the calculation will confirm the umask value : 666-664 = 002

If we want to change the default permission giveb for a file after creation, we need to modify our umask value. For example if we want the following values : rw-rw-rw- We will make the calculation so see what value of umask we should use : 666-666=000. So umask will be : 0000

**umask** 000 : To change the umask

Here we modified the umask value on the user. If we want to change that on a global scale we need to go to as user:

**vim** /etc/profile

In fact, we forgot to explain the role of the first digit here. Here there can be 3 values but each value has a differebt meaning than normal 1,2 and 4

1 → Sticky

2 → SUID

4 → SGID

● Sticky is used to assign the files permissions or directories permissions so that other users except the owner can not delete the files accidentely to prevent some problems.

● SUID works as following : I am root user but I do want to give the access to users to execute the files but what happen here is that I will give the access to them by assigning the SUID to that file. So if I assign a special SUID, whenever a user is executing that files, the file will be executed as the ownership of that file.

● SGID : If we created a directory and 3 users need to access it and do some changes then we need to assign the value 4 so the value of SGID. In that way the users will be in the group of the directory.

# Sudo Access Config

In this part, we will learn how to provide for a user root access without giving them root password so we could execute root commands. In fact, sudo means subtitue user do or super user do.

**cat** /etc/sudoers

*ou*

**visudo** : Now we will give permissions to a particular user

**cat** /dev/null **>** /etc/sudoers : Here we made the values of sudoers completely null. So now when we visudo we will have nothing inside of it.

**visudo** : The file is empty so we will add a user

Host\_Alias SERVERS = localhost, server (Here the server is the @server in root@server)

#User Alias (If I want to make any group)

User\_Alias ADMINS = aravi, user1, user2 (Here the users becomes the ADMINS group)

#Command Alias

Cmnd\_Alias CUSTOM = /sbin/mount, /sbin/fdisk, /sbin/parted

Cmnd\_Alias ADMINSTRATORS **=** /sbin/\*

Cmnd\_Alias CHMOD = /bin/chmod

Cmnd\_Alias CHOWN = /bin/chown

Now we will define some policies so for example if a user typed a wrong command then we should get email alert and other things..

#Defaults

Defaults syslog = auth, insults, syslog\_goodpri=alert

Defaults logfile = /var/log/sudo.log

Defaults timestamp\_timeout = 0, log\_year, tty\_tickets

Defaults mailto = [aravikumar@gmail.com](mailto:aravikumar@gmail.com), mail\_always, mail\_no\_user, mail\_badpass

#Allow users to run commands

root All = (ALL) ALL

aravi All = NOPASSWDl ALL (Here it means no password is needed when running sudo commands so he will be able to execute all commands)

#Group names

%engineers ALL = NOPASSWD: CUSTOM (Here we gave permission to the group ‘engineers’ to all commands in the alias CUSTOM.

%admins ALL = NOPASSWD: ADMINISTRATORS, !CHMOD, !CHOWN (This group can execute all commands defined in the ADMINISTRATORS group except CHMOD and CHOWN,

Now if i switch to aravi user :

**su –** aravi

**sudo cat** /etc/sudoers : The command will work without demanding a password of root to enter.

♣ Instead of each time we create a user we modify /sudoers files, we can directly add the user to the group that we defined in the /sudoers having the right permissions.

# Networking IP

In this section, we will explore the NMCLI tool (Network manager command line interface)

**nmcli device status** : Here we will see the interfaces with its status.

We can create multiple profiles having different configuration for a specific interface so that if I change location and network I can directly activate the corresponding profile. To see the profile used for a connection :

**nmcli connection show**

output : Name UUID TYPE DEVICE

ens33 ffgbe-e23dd.. 802.3\_ethernet ens33

**sudo nmcli connection add type** ethernet **conn-name** home **ifname** ens33 : Now if we execute the command ‘nmcli connection show’ we will see the same output as the ligne before but the field ‘DEVICE’ will be empty because this interface is already used for another connexion.

**nmcli general status** : To see the connectivity running

**nmcli general logging** : To see different protocols, etc…

**nmcli device status** : To see NFC cards

**nmcli device show** ens33 : Show us more details about interface like MAC etc…

**nmcli connection delete** home’

- Now we will configure a connection profile so we will assign an IP Adress, etc…

**nmcli connection modify** home **ipv4.addresses** 192.168.2.141/24 **ipv4.gateway** 192.168.2.2 **ipv4.dns** 192.168.2.2 **+ipv4.dns** 4.4.4.4 **connection.autoconnect** yes **ipv4.method** manual : Here, manual means static ip and do not use DHCP.

**cat** /etc/sysconfig/network-scripts/ifcfg-home : Here we will see all info config about this profile. There are some parameters that will be defined by default.

**nmcli connection up** home

**nmcli connection show --active** : It will show us only active profiles (active connection)

**nmcli device monitor** : To see packets dropping, etc… on specific interface.

- I can give the permissions for a specific user not root to be able to shut down a connection

**nmcli connection modify** aravi **connection.permissions user:**aravi

**nmcli connection down** techarkit : Work from user without root priveleges.

- Now we will learn how we can do all these things without having a lot of commands to write we will use the ‘nmtui’ tool which is a text based user interface.

**nmtui edit** home

**nmtui -hostname** : To directly pass to the hostname in the interface

**nmtui -connect** : To directly pass to the connection section for profiles

# Firewall config

**firewall-config** : It opens a firewall config window. When we launch this windows we will see that there is 2 types of configuration : Runtime config and Permanent config. If we do a runtime config, when we reboot the system, the config will go off.

If I want to use the command line :

**firewall-cmd --help** : To see all rules

**systemctl status** firewall : To see if It is activated.

**firewall-cmd --get-default-zone**

Output : public

**firewall-cmd --set-default-zone =** public

**firewall-cmd --get-active-zones**

**firewall-cmd --zone =** public **--list-interfaces** : To see interfaces added to the public zone

**firewall-cmd --add-interface =** eth0 **--zone =** home: To add interface to a zone

**firewall-cmd --get-services** : To see all services on firewall

We can also make firewall changes by accessing a xml file and modify it :

**cat** /etc/firewalld/zones/public.xml

# SELinux context

Security enhanced Linux is used in addition to a firewall to ensure more secured environement. For example on the edge level of our machine we can secure with firewall and on system level we can secure with SELinux.

In fact everything has labels so files have ones, processes, network ports have labels…

It comes with offering security on multiple levels :

1) Port level security

2) Service level security

3) File level security

In contrast, firewall only provides security on port level. However, with SELinux, le’s say we allocated the port 80 we can still secure a service from this source.

SELinux has 3 modes :

- Enforcing (Enabled)

- Permissive (Not disabled so it will capture logs)

- Disabled

To see if we have SE linux installed we can run the command **sestatus.** I can also use the command **getenforce**.Here we have 2 values of ‘enforcing’. The current mode is the value that it have now so for example let us say I disabled it temporarly and the other entry is how it is configured so after reboot it will have this value. To change the current mode for a temporarly value **setenforce permissive**. If we want it permanently we should change it in the config file

****

**vi /etc/selinux/config** : We should change te entry of **SELINUX=permissive**

The logs of SE Linux is in **/var/log/audit**

**ls -lZ** : To see context of SELinux on a file for example

Output : drwxr-xr-x aravi aravi unconfined\_u : object\_r : user\_home\_t : s0

unconfined\_u : This label is undifined here but here it is the SELinux user based context.

object\_r : Role based access control (RBAC)

user\_home\_t : Desktop (Type of context), so if process or file is for root we will see here admin\_home

s0 : Which level of SELinux is provided.

The value user\_home\_t is not only user or admin. In fact, it can be httpd for example or crond which means that apache has access or cron has access.

Example :

Let’s say we have apache running, and apache will have access to **/var/www/html/** where he will be storing his files and it is SE linux that lets him do that. We can see that configured if we do

**ls -lZ /var/www/html**



So he wee httpd and not user or admin so it means that httpd has access to this directory.

Now let’s say I do not want that Apache saves his files in that directory but instead I want it to be saved on a separate partition on a separate harddrive and we will put read only so in that way if a hacker gets in he won’t be able to modify my web page.

Now if I copy the **/var/www/html** repo to a new repo that we will call it **website** and after that we execute the command **ls -lZ** on the **website**, we will not see the **httpd\_sys\_content** but instead we will see **default\_** which means that Apache will not have access to the **website** repo even if we point him to search for the **index.html** In the **website** repo and not in the **/var/www/html** repo and we can see the logs of that in the **/var/log/audit/audit.log**

Now to fix that I must change the context of that repo so I will use the following command :

**chcon -Rv --type=httpd\_sys\_content\_t /website** : ‘-R’ is used for recursive so it will change files also inside of repo. So now even though Apache had read and write access to the server, SE Linux denied the access because the context type was not correct so now after changing that it must be good to go.

Are these changes persistent ?

If I make a reboot It is persistent but there is a command that restore the context and it will return everything to its default so we should be changing the policy of that.

I will start by restoring con : **restorecon -Rv /website**

**Semanage fcontext -a -t httpd\_sys\_content\_t /website**: So here we are managing the policy of SE Linux. The ‘-a’ let us add a type, the ‘fcontext’ means we are working on files and the ‘-t’ stands for type and we assigned the httpd access to the repo **website**. So now even if we execute the restore context, this repo will have same context access.

If we had an error saying that semanage is not found, we mut install the following :

Sudo yum install policycoreutils-python

**cat** /etc/selinux/targeted/setrans.conf : To see system levels

**sudo sestauts** : To see status of SELinux

**getenforce** : Will return enforcing if it is the case.

**setenforce** 0

**getenforce**

Output : Permissive

**setenforce 1**

**getenforce**

Output : Enforcing

**ls -ldZ** /root/

Output : drwxr-xr-x aravi aravi system\_u : object\_r : admin\_home\_t : s0 /root/

**ls -ldZ** /usr/

Output : drwxr-xr-x aravi aravi system\_u : object\_r : usr\_t : s0 /usr/

**ls -ldZ** /var/

Output : drwxr-xr-x aravi aravi system\_u : object\_r : var\_t : s0 /root/

So here, each directory has its type pf context. So that the service can only access based on the context so if it’s matching.

Now we will take an example with the service httpd

**systemctl start** httpd

**ls -ldZ** /var/www/html/

Output : drwxr-xr-x aravi aravi system\_u : object\_r : httpd\_sys\_content\_t : s0 /var/www/html/

Here, we see that the context is httpd so only a service matching this context will work. So a connection from httpd to a file in this directory will work because the SELinux context is defined as httpd context.

To change the context type :

**chcon** unconfined\_u : object\_r : etc\_t : s0 index.html

**ls -lZ** index.html

Output : drwxr-xr-x root root unconfined\_u: object\_r : etc\_t : s0 index.html

Now, the httpd will not work because the service is different than the context type of the directory. If we set the enforce to 0 he will not put restrictions but will log because it is the mode permissive.

If I want to disable SELinux, I must pass to the file

**vim** /etc/selinux/config

SELinux = disabled

♣ We need to reboot the machine

# SElinux handling ports

**semanage port -l** : This command will show us ports and what is attached to them

So if I want to authorize specific port for Apache for example I can do the following :

**semanage port -a -t httpd\_port\_t -p tcp 8080**

# AppArmor

Either you run SELinux or AppArmor. In fact, SELinux is the default used by RHEL. AppArmor is very similar to SELinux. The main difference is that AppArmor is the default in Debian Systems. But if I want I can switch between them so I can use SELinux instead. AppArmor is more targeted so we create profiles to focus on individual apps so I might have AppArmor that’s only protecting one app which is not really the case with SELinux. SElinux is based on inodes but AppArmor is based on pathnames so AppArmor is more efficient it uses less load on the system but it is as secured as SELinux because it uses path names.

To check if am running AppArmor or not I can execute the following command :

**apparmor\_status** : When we run this command we will se some profiles that are being loaded and then we will see the location of applications that are being protected with AppArmor.

In order to protect an app, we should have a profile and there is the default profile set but first we should verify that we have the following packages : **apt install apparmor apparmor-profiles apparmor-utils**

The AppArmor config files are stored in **/etc/apparmor.d/** and here we will see the same location of apps that we saw when we run the command **apparmor\_status** but they are represented by a . instead of / so for example we saw **/sbin/dhclient.** This is represented with a . **sbin.dhclient** so as a config file inside the **/etc/apparmor.d/** So here we can confirm that AppArmor is based on path names and what it is going to do is give Apps access to limited things in that way if a hacker breaks a web server he will not be able to escalate privileges because AppArmor does not let Apache to do other things than apache things for example.

AppArmor has 2 modes : Enforce and complain (like permissive in SELinux)

AppArmor is more used to protect apps not files.

Normally, we should protect with AppArmor any application that have open ports on the network.

**aa-unconfined** : This command show us the apps that are not protected and listening on a network port.

Enable AppArmor for an application

In the first step, I must verify if it has a profile in the **/etc/apparmor.d/** . Now if the app profile is not there and we do not know the path of the app we can use the **which** command to see where the app is located.

**which apache2**

Now to create the profile we need to have the package **apparmor-utils** so that we do not write all from scratch.

**aa-genprof apache2** : It will generate a profile for the app (But firstly It will enter a monitoring mode for the app, so we will keep that running and while doing so we will do some things with the application like the basic things. After doing that, I return to the terminal where it was moniroting and I can type **S** to scan. When I do that, it will look through the access logs and its gonna show me the things that apache do so that I can choose wether I need those things to be allowed or denied. So it gonna iterate the logs and each time I can either press **A** for allow or **D** for deny… When it is done I can type **S** to save and then **F** to finish and exit. So here it created a profile for that application which is gonna be **usr.sbin.apache2**.

After creating the profile for apache, I should intall a module for apache to let him work with AppArmor (not all apps need that step but apache does need it). But how can I know if I need an extra module or not? In fact we can do a **less usr.sbin.apache2** and we might find an entry saying that we need to do that extra step but sometimes we might not find It so it depends on the Linux distro that it’s been used.

Now I need to activate the profile : **aa-enforce /etc/apparmor.d/usr.sbin.apache2**

If I want to enter complain mode : **aa-complain /etc/apparmor.d/usr.sbin.apache2**

If I want to enter disable mode : **aa-disable /etc/apparmor.d/usr.sbin.apache2**

# firewalld

To know if **firewalld** is active we can use the command **systemctl status firewalld**. In the old versions, **iptables** was used instead of **firewalld** but it is way more complex to configure it. To play with firewall we use the firewall command interface so the following for example :

**firewall-cmd –state**

The way **firewalld** works is that it takes network interfaces and it put them into zones so that we can filter between zones but if we have interfaces in the sam zone we do not filter.

**firewall-cmd --get-zones**

**firewall-cmd --get-default-zones** : All my network interfaces will be put inside of this default zone except if we changed them.

**firewall-cmd --get-active-zones** : To see active zones.

**firewall-cmd --permanent --new-zone=testlab** : We used the **--permanent** to write to disk and not to RAM but we should also write to RAM so we run the same command without this parameter.

**firewall-cmd --reload** : To reload and apply the changes

Now when an interfaces comes online, it has a script that runs and there we could tell which zone this interface must be put in.

# Customize rules of zones with firewalld

We can allow or block traffic coming in or out of interfaces by editing the rules for the zones created on my machine. We must choose the services that we want and **firewalld** comes with a list of these services. **firewall-cmd --get-services**. These services are listed in the directory **/usr/lib/firewalld/services**.

* To allow specific service for a zone we can do the following :

**firewall-cmd --zone=testlab --permanent --add-service=http** : If we do not specify the zone it will take the default zone which is the **public** zone.So here we are allowing the **http** service to the **testlab** zone. Now this is saved to the disk but we should also run the same command without the parameter **--permanent** which will write to RAM and apply the changes in live without the need of reloading all the network service with **systemctl restart network.service**. To test the changes I can do **firewall-cmd --list-services --zone=testlab** (This will show us what it is in RAM, if we want to see what there is in disk we add the parameter)

* To allow specific port to a service we can do the following :

**firewall-cmd --permanent --add-port=8080/tcp**: If we did not put **tcp**, he would add the **tcp** and the **udp** to that port.

**firewall-cmd --permanent --add-port=50000-60000/ucp**: Will add the whole range of ports.

So here instead of editing the XML services files to add ports for example I can directly approve specific ports .

Now to see the ports that are allowed through the firewall we can do the following command :

**firewall-cmd --permanent --list-ports**

# Scheduling Tasks

Here we have the **at** command and the **cron** command where the first one is an old method of scheduling tasks.

# At command

We can use the **atq** command to show the queue of at so the list of scheduled things to be done and we have **atrm** which removes the jobs that we don’t want anymore.

**at 10 PM Fri** : When we hit Enter it will open for us a command prompt where we can enter a command or a sequence of commands to be executed. For example, here I can enter :

**at>/usr/bin/bash /home/dpezet/backup.sh**

In fact, we entered the **/usr/bin/bash** to make the file **backup.sh** executable because it is not.

When we finish entering the commands in the **at** prompt, we can enter ctrl+d.

**at** command is an old command so it may have restrictions on users. In fact we can access the **/etc/** repo and see if we have **either at.deny** or a file called **at.allow**. If we do see them, we will then find in these files the list of users who are allowed or denied to use this command. If we do not see any of these files it means that only admins are allowed to use that command.

Can we schedule recurring jobs with **at**?

The answer is no and that it the limitation of this command, so I can not schedule a job to be done everyday for example, I must instead execute the **at** command everyday which is not recurring at all. Instead we have another command to do that which is **cron**.

# cron command

**cron** uses the **crond** service which is a daemon and he keeps track of time so it watches the time on ur system and it performs several tasks. The complexity of uing **cron** depends on how much picky we are. For example, if we want to schedule a job but we don’t care at what exact time I want to execute it so like weekly or daily or monthly, we find in the **/etc/** several repo like **cron.daily**, **cron.hourly**, **cron.weekly**… that we can access and configure inside of them what we want to do. So we can write scripts inside of these repos that will be executed in a recurrent way so monthly or weekly…

Now if we want to schedule a job at a very exact time, here we need to configure a fully described job. These jobs are specified in the **crontab** text file in **/etc**. Some distros have a repo **cron.d** where there will be the several **crontab** text files with the jobs scheduled .

# Regular users scheduling jobs

In fact, the admin must add a cronjob entry by adding the name of the user. We can see the documentation in **/etc/crontab**. However, here we need the admin privilege so it is not that practical. However, they can have their cron jobs in their home directory. To see these files it is either hidden in the home directory and other distro put it in the **/var/spool** repo. But regular users are not allowed to edit these files, only admins. If they want to edit them, they must use the **crontab** command to edit their own jobs. So I can use **crontab -e** to edit and we will get the editor to put there the jobs. After adding the jobs, I can see the different jobs with the **crontab -l**. Now where is that stored? So either in home directory like we said before or in the **/var/spool**

Now as an administrator, I can edit jobs for different users while using parameters with the **crontab** command.

**crontab -e -u elie**

In fact I could put them in the **/etc/cron.d/** for the specific users but they won’t see them and will not be able to control them. The advantage of using the **crontab** command is that users will be more autonome so they will have their jobs in the home directory or **/var/spool** without needing the admin privilege.

# anacron command

Many distros does not ship anacron by default (also called **cronie**).

If we scheduled a job with **cron** but the machine was in sleep mode at the time of the scheduled job, when we fire up the machine, the job will b skipped because the time has already passed. However, with **anacron**, it does not skip the job so it will do it even when we fire up the machine after the time scheduled for the job has passed.

If we access the **/etc/anacron** we will see the **cron.daily**, **cron.weekly** that are in the **/etc** for which we said that they are not scheduled but actually it is **anacron** that schedule those.

# git command

**git checkout -b newBranch**: This command will create a new branch so other than the master branch and will be committing changes to thi new branch.

**git merge newBranch** : Will update changes from the branch that im a present in to the branch called ‘newBranch’.

# Backup & Restore

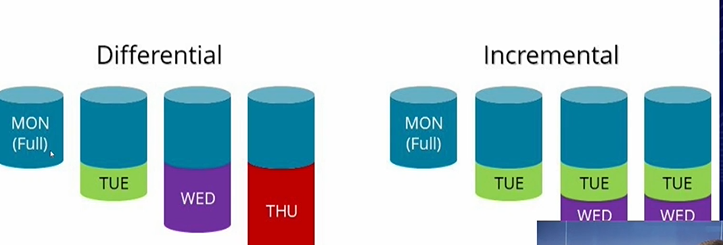
-We can archive our things with **tar** which means tape archive.

**tar cvzf backupFile /home/elisaba** : The ‘c’ is used to add files like compres them, the ‘v’ for verbose, the ‘z’ is used to make it use Gzip to compress and the ‘f’ is to choose the files. Here we are archiving my home directory into the file called ‘backupFile’.

**-**Now to extract : **tar xvzf backupFile**

# Backup & Restore / Advanced

Difference between differential and incremental backup :



With differential method, each day we use the full backup as a baselin and just adds what is new (But when we add the new modifications we will not have really a backup containing the modification of today and the full backup because the fullbackup that we see other than the real first backup is just like a pointer for the real full backup)This is going to get bigger and bigger day because each day, the first full backup is getting older and missing a lot of new changes. The downside is that when I want to make a restore, I should restore the full backup and the newest backup because as we said, the newest backup does not contain the real first full backup of Monday. For a long term, we can see that the last backup will be so big which makes it sometimes impossible to really backup such data that’s why we will use instead something called incremental backup.

With incremental method, we begin by doing a full backup, by instead of always having the first full backup as a baseline and making a backup by comparing what is new with the first full backup which is the case with differential, however we will be based on the backup just before and adds the changes by comparing with that which will let us have smaller backups. The downside is that for example if I want to restore the backup on Friday, I should restore the full backup, then restore Tuesday, Wednesday and Thursday to make it. And if I lost for example the Tuesday backup we will be finishing with missing data. So what big enterprises do, is make a lot of incremental backups throughout the day in that way we will lose the smallst amount of data if we lose a specific incremental backup.

**dar** (disk archive) is a newer utility than **tar**. In fact, **tar** is not that intelligent so it copies what we tell it to copy and that’s all.

In fact, **dar** works a lot like **tar** but u can tell it to only backup changes for example from the previous backup and it supports Differential and incremental type of backups.

**dar -R /home/elie -c /mnt/FullBackupFile** : The ‘R’ means recursive and the ‘c’ means destination.

Now we will make some changes in our home directory so we want to let **dar** backup that and we said that we have 2 methods (differential and incremental)

Incremental

**dar -R /home/elie -c /mnt/IncrementalFile1 -A /mnt/FullBackupFile** : So the ‘-A’ let us point to the full backup. So here we used incremental backup which means that the new backup file called ‘IncrementalFile’ is so small and gonna contain only the changed data so we will always need the Full first backup to build from there.

If I want to make a new Incremental backup I must point to the last incremental backup

**dar -R /home/elie -c /mnt/IncrementalFile2 -A /mnt/ IncrementalFile1**

Differential

It is always based on the last full backup.

**dar -R /home/elie -c /mnt/DifferentialFile -A /mnt/FullBackupFile**

So here if I want to add a new differential backup i should point on the last full backup

Restore the full backup

**dar -x /mnt/FullBackupFile**

Restore the incremental

**dar -x /mnt/ IncrementalFile1 -w**

**dar -x /mnt/ IncrementalFile2 -w**

So here I should pass by every incremental and then get to the first full backup

# Copy and convert disk

**dd** command : Which mean copy and convert.

Let’s say I want to take a disk like **/dev/sdb1** I can grab it and convert to another disk or to an image for example.

**dd if=/dev/sdb1 of=/dev/sdd1** : ‘if’ means input file so the file that we want to convert and ‘of’ is the output file so the destination. So here it is cloning the whole disk (It takes a time because it is going to read all blocks of the disk even those that are empty so when it copies its gonna be bigger because it will also copy the blank disks)

# YUM Repository

(1) AppStream : Used for installing the applications

(2) BaseOS : Used for installing Base OS required packages

(3) YUM Repository : Stores both of AppStream and BaseOS repositories

**yum install** httpd

# NTP Server and Client

Send timestamp

Receive timestamp(UDP)

Client Server

It is mandatory that the time is synchronized between server and client

● Commands to execute on NTP Server side :

**dnf install chrony**

**systemctl enable chronyd**

**vi** /etc/chrony.conf

**allow** 192.168.45.0/24

**systemctl restart chronyd**

**firewall-cmd --permanent --add-service =** ntp

**firewall-cmd -reload**

**ntpupdate <**NTP SERVER ADDRESS**>**

● Commands to execute on NTP Client side :

**dnf install chrony**

**systemctl enable chronyd**

**Server <**NTP SERVER ADDRESS**>**

**vi** /etc/chrony.conf

**server** 192.168.175.128

**systemctl restart chronyd**

# Scheduling jobs

**cat** /etc/crontab : To see the entries description

Now we will make an example of doing a job every 1 minute (A script was written to be executed)

**crontab -e** : Will open a file to edit

\* \* \* \* \* : This means it will be executed everytime (Every minute, houre, everyday..)

\* \* \* \* \* **sh** /root/binbash/quotes.sh

So here we are saying to execute this shell everytime. Now to check if that thing worked, we need to check the log file where we can see the last time the shell was executed.

**cat** /var/log/cron **| grep** quotes

Another example :

● If I want to execute it every 5 minutes :

\*/5 \* \* \* \* **sh** /root/binbash/quotes.sh

● If I want to execute it between 2 values I use (-)

\*/5 21-23 \* \* \* **sh** /root/binbash/quotes.sh

So here we are executing every 5 mintes between hour 21 and 23

● If I want to execute it between at values I use (-)

\*/5 21,23 \* \* \* **sh** /root/binbash/quotes.sh

So here we are executing every 5 mintes between at hour 21 and 23

**crontab -l** : To see the entries I added

# Rsync

Copy locally :

Rsync is a tool used to copy data from source to destination which can be from a server to another or from a directory to another on the same server.

**rsync -av /var/log /mnt/backup** : The ‘-v’ stands for verbose and the ‘-a’ stands for archeive mode which means that It’s gonna copy a complete directory recursively so with it’s all of its subdirectories and it’s gonna retain the permissions etc..

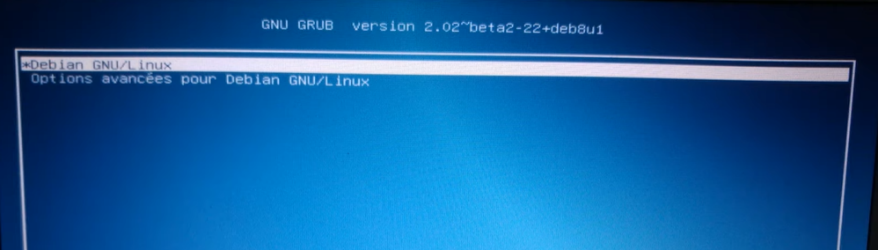
**rsync -av --dry-run /var/log /mnt/backup** : The ‘--dry-run’ will simulate the copy so it will show us output with the ‘-v’ like it has done the copy but actually it did not do anything. It’s a way to see what will it do if we really execute the command.

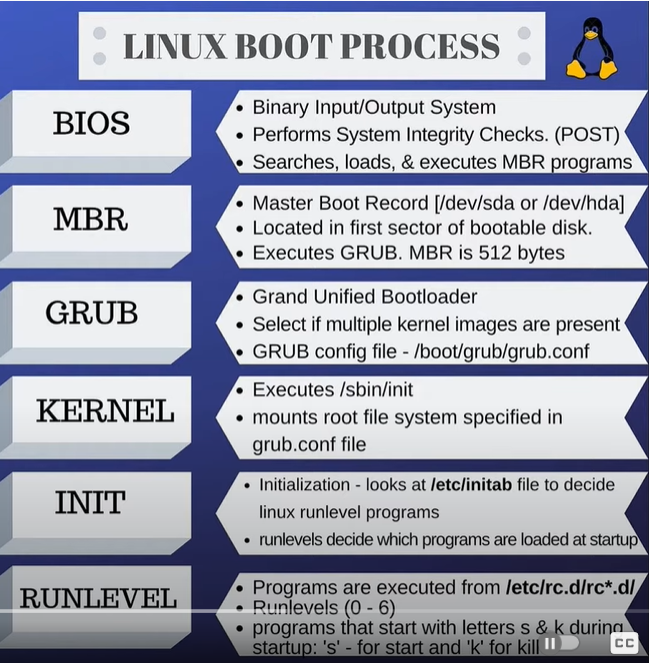
Copy remotely :

Rsync must be installed on both of the servers where we want to copy the files.

**rsync -av /var/log root@139.177.10.10:/mnt/backup**

# Grub (Grand Unified Bootloader)





# Systemd

It is the daemon so services that run in the background.

Systemd is fired in the INIT phase in the photo above. Majority of linux distro uses systemd by default.

Services are defined in **/lib/systemd/system** (In fact, we should not touch these files here if we want to make some modifications but instead we should go the **/etc/systemd/system**) So here it will not edit anything in the /lib but everything will be edited in the /etc

If we have httpd on our machine we will definitely have httpd service that is executed and found in the /etc/systemd/system

# SysVinit

Like systemd and uses a simplified approach to service startup.

# chkconfig

This command let us run services on a specific run levels.

**chkconfig --list**

**chkconfig --level 235 mysqld on** : So here we are running the service mysqld on run levels 2 3 and 5

Example :

**Systemctl start httpd**

**Systemctl start httpd.service**

So here actually when we launch the **httpd** it will search for the **httpd.service** uni file.

# lsmod

**lsmod | grep less**: This command is used to list modules.

**modinfo** name\_of\_module : Here we will see more info about the module.

# modprob

Add modules to system while kernel is running

# What is the impact of running the dash shell instead of a bash shell?

A small percentage of the commands will be different. Dash shell is faster.

# GPG (Packages checking of repos)

rpm --import /etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release

[13:41] BOU SABA, ELIE

J'ai récupéré la clé GPG qui se trouvait dans :

/etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-8

Je l'ai copié donc de mon poc14 vers le bon endroit sur ma VM et il fallait modifier les fichiers de repo en ajoutant l'entrée suivante pour pointer vers la destination de la clé :

gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-8

**Implementation de KVM**

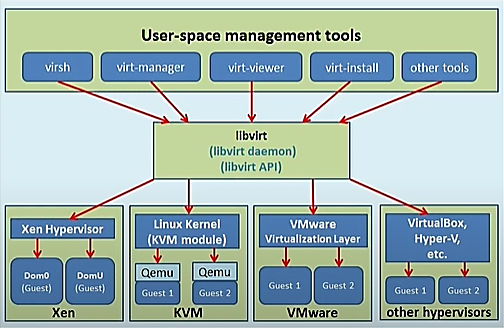
# Definition de KVM

KVM (Kernel-based Virtual machine) permet la virtualization du Linux Kernel et donc avoir un hyperviseur de type 1.

KVM est extensible, non rattaché à un fourniseur (propriétaire Red hat).

KVM utilise libvirt(virsh) qui est une bibiliothèque de virtualisation qui permet la communication avec le hardware.

Il faut pas oublier d’activer la fonctionnalité de virtualisation au niveau hardware du processeur pour pouvoir mettre en place la VM.



QEMU est un émulateur de hardware utilisé par KVM.

# Vérifier si la virtualization du processeur est activée

cat /proc/cpuinfo | egrep "vmx|svm"

Sinon vérifier si on peut déjà installer KVM avec **sudo yum install -y cpu-ckecker** et exécuter ensuite la commande **kvm-ok**

# Installer les packages KVM et activer les services nécessaires

**yum install qemu-kvm libvirt libvirt-python libguestfs-tools virt-install -y**

**systemctl enable libvirtd**

**systemctl start libvirtd**

# Comprendre les allocations stockage d’une VM

**virt-install --name VM-NETBOX --memory 4000 --vcpus 4 --disk size=50 --os-type linux --os-variant rhel8.5 --location /iso/rhel-8.5-x86\_64-dvd.iso --graphics none --noreboot --initrd-inject /root/VM-NETBOX.ks --extra-args="ks=file:/VM-NETBOX.ks console=ttyS0 console=ttyS0,115200n8"**

Cette méthode aboutit à la création d’une VM en utilisant un fichier Kickstart qui définit quelque paramètre. En fait, la VM quand elle sera crée, une image dans /var/lib/libvirt/images/ sera crée ayant une extension qcow2. C’est le format par défaut de stockage des disques virtuelles.

# Utiliser les VM

1. Pour afficher les VMs présent sur notre machine : **virsh list --all**
2. Plus d’informations sur une VM (domain info) : **virsh dominfo testVM**
3. Eteindre une VM : **virsh shutdown testVM**
4. Démarrer une VM : **virsh start testVM**
5. Auto démarrage d’une VM : **virsh autostart testVM**
6. Accéder la console de la VM : **virsh console testVM**
7. Clonage d’une VM : **virt-clone --connect qemu:///system --original testVM --name testVM2 -f /MountTest/**

# Installation d’une machine virtuelle KVM sur rhel 8

1. Installation des modules nécessaires :

**yum module install virt**

**yum install virt-install**

1. Démarrage du service libvirtd :

**systemctl start libvirtd**

1. Vérification si le system est prêt a créer des VMs :

**virt-host-validate**

1. Création d’une VM :

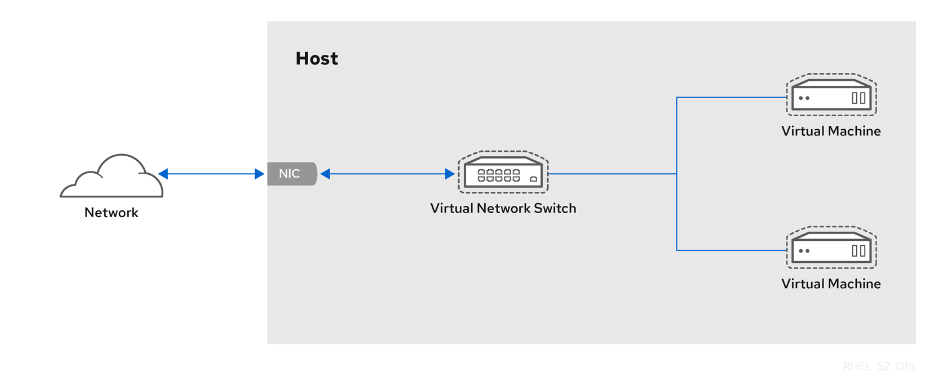
**virt-install --name VM-NETBOX --memory 4000 --vcpus 4 --disk size=60 --os-variant rhel8.2 --cdrom iso/RHEL-8.2.0-20200404.0-x86\_64-dvd1.iso**

1. Démarrage d’une VM et activer l’autostart:

**virsh start VM-NETBOX**

**virsh autostart VM-NETBOX**

1. Comprendre le réseau virtuel



Le réseau dont la VM est placé attribue des adresses IP en utilisant **dnsmasq** généré par le service **libvirt**. Par défaut, toutes les machines virtuelles crées sont mis dans le même réseau ‘default’. Ces VM peuvent accéder au réseau sur lequel la machine physique est lié et le réseau extérieur mais ces VMs ne sont pas visibles par le réseau extérieur. Pour cette raison, des règles NAT doivent être ajoutés pour résoudre ce problème

L’interface **virbr0** est créé par défaut par le service **libvirtd** et c’est le point d’accès aux VMs dans le réseau virtuel. On peut voir l’IP attribué à cette interface :

**ip addr show virbr0**

1. Connection console sur la VM:

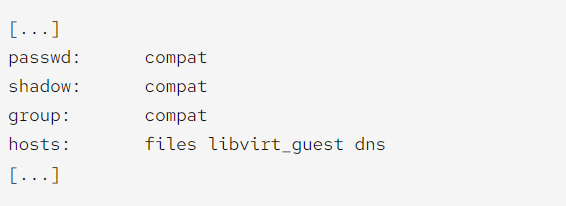
**virsh console VM-NETBOX --safe**

1. Connection SSH sur la VM:

* Installation du package **libvirt-nss**

**yum install libvirt-nss**

* Modification du fichier de config /etc/nsswitch.conf et ajout de **libvirt\_guest** à la ligne **hosts**.



* Se connecter à la VM en utilisant son nom :

**ssh root@VM-NETBOX**

1. Migration d’une VM d’un serveur à un autre sans shared storage

1. Stop you VMs: virsh shutdown VM\_MACHINE and virsh autostart VM\_MACHINE --disable

2. Create a configuration xml dump: virsh dumpxml VM\_MACHINE > VM\_MACHINE\_dump.xml

3. [cp](https://en.wikiversity.org/wiki/Linux/Basic_commands/cp) files to new server image directory: /var/lib/libvirt/images or virsh vol-list POOL\_NAME --details (verify files owner and group)

4. Depending on your Origin and destination host you may need to edit xml file and modify network interface and remove [MAC](https://en.wikiversity.org/w/index.php?title=MAC&action=edit&redlink=1) address. (See [MacVTap](https://en.wikiversity.org/wiki/KVM/MacVTap" \o "KVM/MacVTap))

5. Create machine but do not start it: virsh define VM\_MACHINE\_dump.xml (virsh list --all)

6. Configure autostart: virsh autostart VM\_MACHINE

7. Start Machine: virsh start VM\_MACHINE

**scp VM-NETBOX\_dump.xml** [**bousabae@172.16.118.128:/home\_nfs/bousabae**](mailto:bousabae@172.16.118.128:/home_nfs/bousabae)

**scp VM-NETBOX-4.qcow2** [**bousabae@172.16.118.128:/home\_nfs/bousabae**](mailto:bousabae@172.16.118.128:/home_nfs/bousabae)

Faire attention il faut modifier le fichier xml si l’erreur machine type se lance en essayant de lançer la VM ce qui signifie que le qemu avec lequel la VM à été crée n’est pas présent sur la machine destinataire donc il faut voir les qemu supportés sur la machine destinataire avec la commande /**usr/libexec/qemu-kvm -machine help** et modifier la ligne machine type du fichier xml ensuite lancer la VM de nouveau.

# Migration de python vers 3.8

* Python est primordiale pour installer netbox. Débutons par l’installation des librairies nécessaires pour Python :

**dnf install gcc openssl-devel bzip2-devel libffi-devel**

* Téléchargement et installation de Python3.9 :

**cd /opt**

**wget** [**https://www.python.org/ftp/python/3.8.8/Python-3.8.8.tgz**](https://www.python.org/ftp/python/3.8.8/Python-3.8.8.tgz) **--no-check-certificate**

**tar xzf Python-3.8.8.tgz**

* Après l’extraction, il faut configurer les fichiers sources et compiler Python :

**cd Python-3.8.8**

**./configure --enable-optimizations**

* Ensuite, il faut empêcher le remplacement du fichier binaire par defaut de Python et donc créer un fichier binaire séparé.
* **Yum install make**

**make altinstall**

* Installation des librairies restants :

**yum install -y gcc libxml2-devel libxslt-devel libffi-devel libpq-devel openssl-devel redhat-rpm-config**

* Ajouter la version de python3.8 au système

**update-alternatives --install /usr/bin/python3 python3 /usr/local/bin/python3.8 1**

* Utiliser la version 3.8 à la place de 3.6 (on peut voir les versions dispo sur notre système avec la commande **update-alternatives –config)** pour la commande python et python3, donc il faut configurer python3 pour utiliser la version 3.8

**update-alternatives --config python3**

Ensuite, faire un lien de python à python3 et donc forcement a python3.8

**update-alternatives --config python**

* Vérifier que la bonne version de Python est installée :

**python3 -V**

# Slurm :

## Definition:

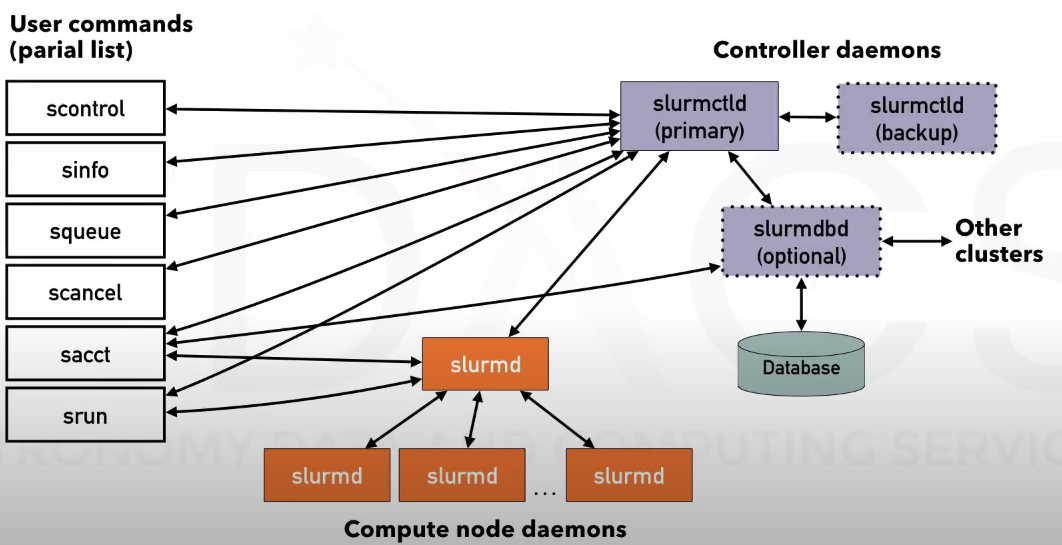
* Gestion des Clusters et planificateur des taches qui s’exécutent sur le cluster.
* Un planificateur des taches est nécessaire dans le but de maximiser l’utilisation du système de clusters d’une manière idéale pour les utilisateurs.

## Fonctionnalités :

* Allocation d’accès aux ressources (compute nodes) aux utilisateurs.
* Framework pour le démarrage, l’exécution et la surveillance d’une manière parallèle sur les nodes choisis.
* Gestion d’une file d’attente de taches à exécuter.

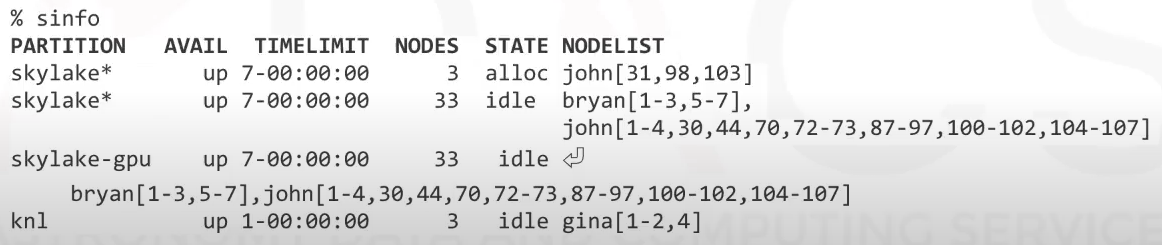
## Architecture :

* Consiste d’un daemon **slurmd** qui est lancé sur chaque compute node.
* Consiste d’un daemon **slurmctld** qui est lancé sur le management node.



## Acquérir d’informations :

* **sinfo** commande : Montre les ressources offertes par un cluster. Par défaut cette commande montre aussi une liste des partitions disponibles. Une partition est un set des compute nodes qui sont groupés ou combinés logiquement.

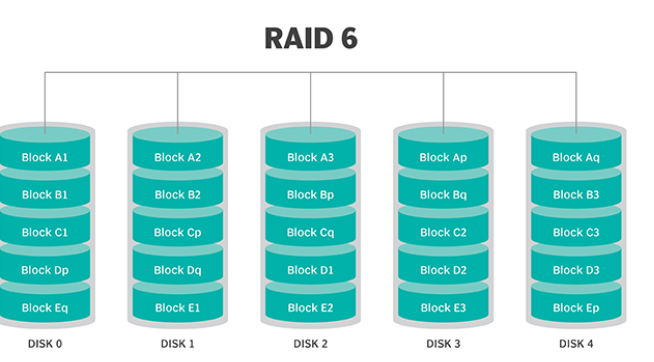
****

**sinfo -N -l** command : **-N** signifie ‘Node oriented’ et **-l** pour avoir plus d’info sur les nodes (Nombre de CPUs, Mémoire, disk temporaire…)

* **squeue** commande : Montre les ressources sont alloués aux différentes tâches.

# Stockage

RAID 6 :



Avantages :

* RAID6 est plutôt utilisé dans l’archivage car c’est trop tolérant au perte des disks.
* RAID6 utilise 2 disks qui joueront le rôle des disks de parités donc si on avait 4 disks en total dans notre infrastructure, la moitié des disks seront utilisés pour stocker les données et l’autre moitié pour stocker les bits de parités. Donc le pourcentage des disks qu’on peut utiliser pour stocker les données augmente si on ajoute plus de disks

Inconvénients :

* Lenteur d’écriture sur les disks : Chaque set de parité doit être calculé séparément en utilisant RAID6 ce qui a aussi un impact sur la vitesse avec laquelle on reconstruit notre baie de stockage après l’échec d’un disk.
* Besoin toujours de 2 Disks de parités ce qui est couteux.
* Besoin d’un controlleur donc un hardware special pour supporter cette technologie.

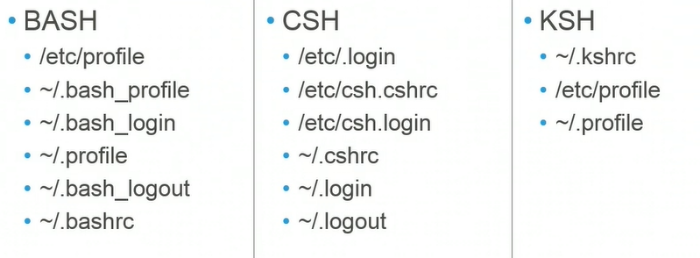
**Bash scripting**

# Type of shells

Bourne shells

* Bourne-again Shell (bash)
* Korn Shell (ksh)
* Z shell (zsh)

Some configuration files exists and runs by default when we log in into a user sessions and these config files can have different names on different shells. So, for example :



The **/etc/profile** is a script that runs for all users but the hidden script which is the **bash\_profile** is used for a specific user and only executed when switching to this user. So here when we switch to a specific user, the **/etc/profile** script will be executed first and then followed by the **bash\_profile** script

# Introduction to bash scripting

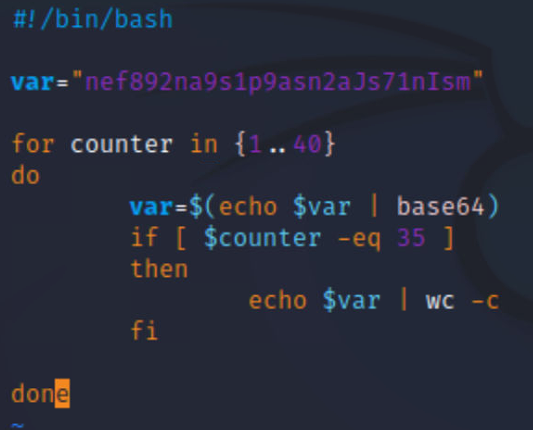
- When we executes a script, we are not launching a new process but it is the role of the interpreter which is ‘bash’ to execute the script. There are 3 ways to execute a bash script

**./**script.sh

**bash** script.sh

**sh** script.sh

Example of a bash script :



# Special variables

- In a script the following operators means the following :

**$#** : Number of arguments

**$@** : Get list of arguments

**$0** : Name of actual shell script

**$1** : First argument passed when calling the script

**$2** : Second argument passed when calling the script

**$$** : Process ID of currently executing process

**$?** : Exit status of the last executed process (parent process which is the script or child processes which are functions for example In the script)

# Variables

When assigning a value to a variable we call the variable without the ‘$’ sign. However, we use the ‘$’ sign to allow this variable’s corresponding value to be used in other code sections.

**domain=$1**

with bash, all contents of the variables are treated as string characters.

♣ We do not put a space between the name of variable and the value.

# Arrays

Array values are indexed beginning with 0. To declare a array with some values :

**domain=(elie roy marc)** // Here we have 3 values in the array

**echo ${domain[0]}**

**domain=(“elie roy” marc)** // Here we have 2 values in the array (we can also use single quotes it’s the same as double quotes) So the quotes means that it is the same value even if there is space.

# Compare variables

Comapring strings :

|  |  |
| --- | --- |
| == | is equal to |
| != | is not equal to |
| < | is less than in ASCII alphabetical order |
| > | is greater than in ASCII alphabetical order |
| -z | if the string is empty (null) |
| -n | if the string is not null |

We should note that here the variable that we want to compare it must be put in “ “ in that way we are telling bash to handle the variable as a script.

if [ "$1" != "HackTheBox" ]

then

echo -e "You need to give 'HackTheBox' as argument."

exit 1

elif [ $# -gt 1 ]

then

echo -e "Too many arguments given."

exit 1

if [[ $# > 1 ]]

♣ So here we used the double brackets to compare strings

Comapring integers :

|  |  |
| --- | --- |
| -eq | is equal to |
| -ne | is not equal to |
| -lt | is less than |
| -le | is less than or equal to |
| -gt | is greater than |
| -ge | is greater than or equal to |

Comapring files :

|  |  |
| --- | --- |
| -e | if the file exist |
| -f | tests if it is a file |
| -d | tests if it is a directory |
| -L | tests if it is if a symbolic link |
| -N | checks if the file was modified after it was last read |
| -O | if the current user owns the file |
| -G | if the file’s group id matches the current user’s |
| -s | tests if the file has a size greater than 0 |
| -r | tests if the file has read permission |
| -w | tests if the file has write permission |
| -x | tests if the file has execute permission |

**#!/bin/bash**

# Check if the specified file exists

if [ -e "$1" ]

then

echo -e "The file exists."

exit 0

else

echo -e "The file does not exist."

exit 2

fi

Logical operators :

|  |  |
| --- | --- |
| ! | logical negotation NOT |
| && | logical AND |
| || | logical OR |

# Example

**#!/bin/bash**

# Check if the specified file exists and if we have read permissions

if [[ -e "$1" && -r "$1" ]]

then

echo -e "We can read the file that has been specified."

exit 0

elif [[ ! -e "$1" ]]

then

echo -e "The specified file does not exist."

exit 2

elif [[ -e "$1" && ! -r "$1" ]]

then

echo -e "We don't have read permission for this file."

exit 1

else

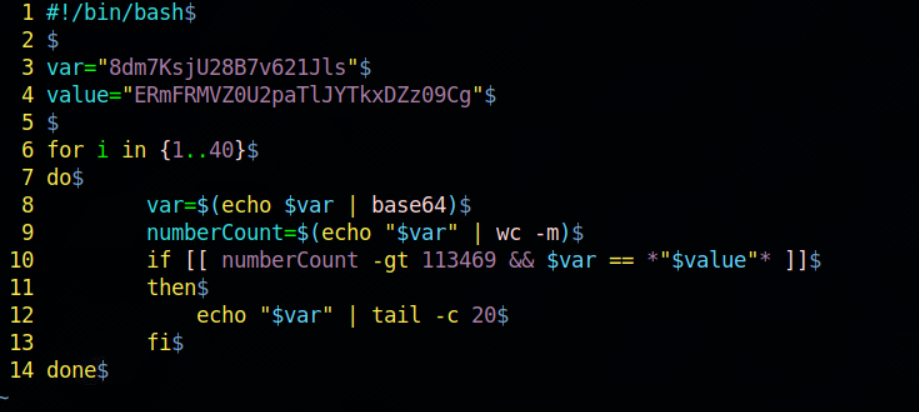
echo -e "Error occured."

exit 5

fi

# Exercice

Create an "If-Else" condition in the "For"-Loop that checks if the variable named "var" contains the contents of the variable named "value". Additionally, the variable "var" must contain more than 113,469 characters. If these conditions are met, the script must then print the last 20 characters of the variable "var". Submit these last 20 characters as the answer.



# Arithmetic expression

**#!/bin/bash**

increase=1

decrease=1

echo "Addition: 10 + 10 = $((10 + 10))"

echo "Substraction: 10 - 10 = $((10 - 10))"

echo "Multiplication: 10 \* 10 = $((10 \* 10))"

echo "Division: 10 / 10 = $((10 / 10))"

echo "Modulus: 10 % 4 = $((10 % 4))"

((increase++))

echo "Increase Variable: $increase"

((decrease--))

echo "Decrease Variable: $decrease"

# Count number of characters

**#!/bin/bash**

htb="HackTheBox"

echo ${#htb}

# Input/Output

echo -e "Additional options available:"

echo -e "\t1) Identify the corresponding network range of target domain."

echo -e "\t2) Ping discovered hosts."

echo -e "\t3) All checks."

echo -e "\t\*) Exit.\n"

read -p "Select your option: " opt

case $opt in

"1") network\_range ;;

"2") ping\_host ;;

"3") network\_range && ping\_host ;;

"\*") exit 0 ;;

esac

Here, the ‘-p’ options insures that our input stays on the same line. The value entered by the user will be put in the variable ‘opt’. After that, depending on the value entered we will be calling a function.

# Output control

In fact, we learned about redirecting output to a file but we can use the **tee** command which will do 2 things in a time: It will output and then redirect to the file.

Examples :

df -h | tee disk\_usage.txt

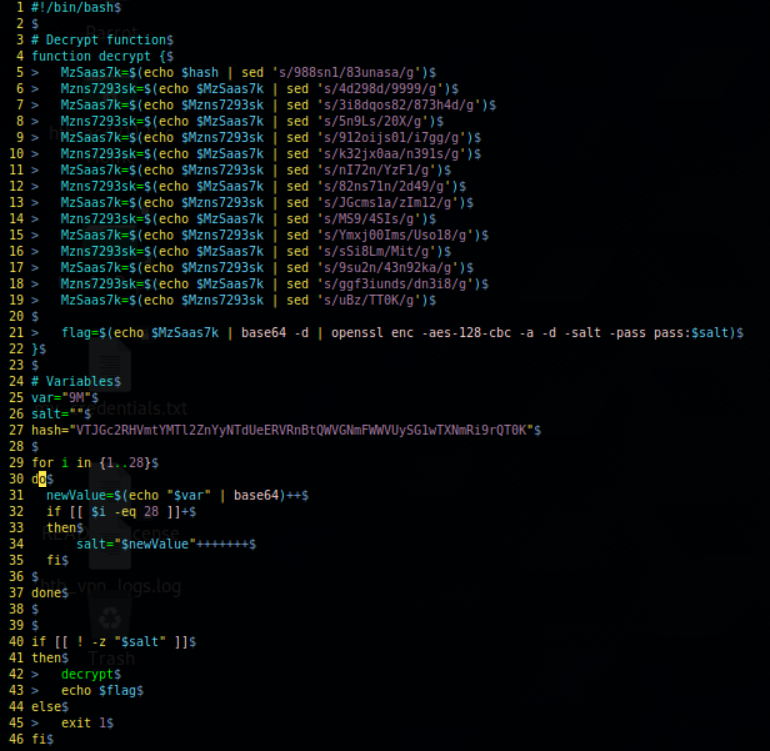
Here, the output given by the **df** command will be redirected and written to the file.

command | tee file1.out file2.out file3.out

We can use the tee -a option to append and not override the file.

# Exercice

Create a "For" loop that encodes the variable "var" 28 times in "base64". The number of characters in the 28th hash is the value that must be assigned to the "salt" variable.



# Difference between if-else and switch case

With **switch case** we are comparing to a specific value so we can not have cases like greater than etc… but with the **if-else** statement we can have more flexibility with our use cases.

case $var in

value1 ) function1 ;;

value2 ) function2 ;;

value3 ) function3 ;;

esac

# Functions

Functions must always be declared in the beginning of our script so that it will be defined in our program because the script is processed from top to bottom. There are 2 ways to define a function:

#### Method 1

function name {

<commands>

}

#### Method 2

name() {

<commands>

}

The function is called only by calling the specified name of the function.

# Parameter passing

With functions, parameters passing works the same way as passing parameters to a shell script. So we use the **$1** till **$n**

♣ With bash scripting unlike other programming languages, all variables are processed as globally declared variables unless declared with the **declare**, in this case they will be local to the present block.

**#!/bin/bash**

function print\_pars {

echo $1 $2 $3

}

one="First parameter"

two="Second parameter"

three="Third parameter"

print\_pars "$one" "$two" "$three"

# Return status of functions

When we start a new parent process for example launching a script containing functions that will be executed as child processes, each child process will return a value or a status for its parent which is the script. In fact, we can visualize this value with the **$?** to read the return code.

function given\_args {

if [ $# -lt 1 ]

then

echo -e "Number of arguments: $#"

return 1

else

echo -e "Number of arguments: $#"

return 0

fi

}

given\_args

echo -e "Function status code: $?\n"

given\_args "argument"

echo -e "Function status code: $?\n"

# Debugging

Bash allows us to debug our code with the **-x** (xtrace) or **-v** options.