Linux Basics for Hackers - Notes

1) BINARIES

Located in -> /usr/bin and/or /usr/sbin

2) LINUX FILE SYSTEM

```
'/' -> The actual root system. The top most.
```

Inside it ->
`/boot` - Kernel image
`/home` - user dir
`/proc` - view of internal kernal data
`/dev` - special device files
`/sbin` - binaries
`/root` - SuperUser's Home Dir (different from '/')
`/etc` - Sys config
`/mnt` - GenPurpose Mount point
`/sys` - Kernel's view of HW
`/bin` - also binaries
`/lib` - libraries
`/usr` -> /sbin, /bin, /lib (more of the same stuff) /media - for eject-able media

3) cd Command

use `$ > cd ..` to move up 1 level
`$ > cd .. ..` for 2 levels & `$ > cd .. .. ..` for 3 levels and so on

4) Search-based commands

`$ > locate aircrack-ng` -> finds all occurrences.
`$ > whereis aircrack-ng` -> finds all BINARIES of the target passed (usually with man pages).
`$ > which aircrack-ng` -> finds the binary file located in the PATH variable of the system.
$ > find -directory -option -targetExp -> finds literally everything.
Eg: `$ > find /etc -type f -name apache2`
-- altho apache2.* will find all file extensions but first name as apache2
Lastly, grep to filter

`$ > ps aux | grep apache2` -> will filter from all auxilliary processes containing apache2

5) 'cat' is versatile

`$ > cat _file-name_` -> will spill the file contents.
`$ > cat > _file-name_` -> will let you write in it BUT WILL REPLACE ALL EXISTING DATA.
`$ > cat >> _file-name_` -> will actually let you append the text you enter.

6) Renaming doesn't exist in Linux

So we use
`$ > mv newfile newfile2` -> to essentially rename the file

7) REMOVING A DIRECTORY

Only the directory : `$ > rmdir`
When that fails : `$ > rm -r` (recursively delete everything in it)

8) TEXT MANIPULATION

'head' and 'tail' :
`$ > head _file_name_` -> first 10 lines
`$ > head -n _file_name_` -> n number of lines from the start
`$ > tail` -> for bottom lines (+ specialized with a count)

`$ > nl _path_to_file_or_file_name_` -> will number all the lines.
// 'cat' can & should be clubbed with 'grep' as and when needed.
Eg: `$ > cat snort.conf | grep output`

Sed command - for find and replace ->
`$ > sed s/search_term/replace_term/g path_to_filename > newfile_name`
-> 's/' will find the term, '/g' is for replacing globally. Rest is elementary.
-> Removing the '/g' will only replace the first occurrence.
Adding a number there can limit the number of occurrences to be changed. '/3' will only replace the first 3.
Eg: **`$ > sed s/mysql/MySQL/g /etc/snort/snort.conf > snort2.conf`**

`$ > more _file-name_` -> offers a scroll-able page if the file is to big.
`$ > less _file-name_` -> "less is more" -- offers a filter to search for the term should you need to -- use the '/' key. Still scroll-able but with better functionality.

9) NETWORKING

'loopback' addr -- same as 'localhost' = 127.0.0.1
`$ > iwconfig` -> check for wireless adapter info -- good for getting power, the mode [monitor, managed, promiscuous] etc.

Changing IP Addr :
`$ > ifconfig eth0 192.1688.181.155`
-> modifies what your router sees to redirect packets

Changing Netmask +/ Broadcast :
`$ > ifconfig eth0 192.168.181.155 netmask 255.255.0.0 broadcast 192.168.1.255`
-> Netmask is the subnet mask -- determines the portion of the IP Addr to the NW and which refers to the host. Here, first 2 octets (16 bits) represent the network and last 2 show the hosts within that network.
-> Broadcast is the addr used to send packets to all hosts on the same network segment. Default (bcz of subnet) would become 192.168.255.255 but here overridden to 192.169.1.155 -- Basically packets sent to this will be broad-casted on that specific sub-network.

MAC Spoofing :
Take down the interface, change the Addr, restart
`$ > ifconfig eth0 down`
`$ > ifconfig eth0 hw ether 00:11:22:33:44:55`
`$ > ifconfig eth0 up`

Assigning new IP via DHCP Server :
Server runs of '*dhcpd*' - the daemon. Requested via '*dhclient*'. Requires a DHCP assigned IP addr. (Note : 'dhclient' is for Debian, will vary in other distros.)
`$ > dhclient eth0`
-> Here, DHCPDISCOVER req is sent by this command and then receives an offer from the DHCP i.e DHCPOFFER. Now, ifconfig will show a difft IP addr as given by the DHCP Server.

Manipulating DNS :
Use 'dig' :
Directly pass-on the domain and add the 'ns' tag to make the domain as the nameserver itself. 'mx' will fetch the mail-exchange server.
`$ > dig hackerarise.com ns`
OR
`$ > dig hackerarise.com mx`
-- Some Linux servers use BIND (*Berkeley Internet Name Domain*) which is just a fancy name for DNS.

Changing your DNS Server :
Edit the file '/etc/resolv.conf'. There you will see the domain, search & nameserver fields. Swap the values here to switch your DNS server.
Other method to do the same (although this cleanly overwrites the file's content) -> `$ > echo "nameserver 8.8.8.8" > /etc/resolv.conf`

Mapping your own IP Addrs :
'hosts' file located in **/etc/hosts**. Useful for hijacking a TCP connection on your LAN to direct traffic to a malicious webserver by using a tool like *'dnsspoof'*
Usually this file has a mapping for your localhost only BUT you can map any website to any IP Address. Eg : "192.168.181.131 bankofamerica.com". Decent for local network attacks. [although 'dnsspoof' & 'Ettercap' can be used]

10) HANDLING SW PACKAGES

Search for package in local repo:
`$ > apt-cache search *keyword*`
-> Eg: `$ > apt-cache search snort`

Install, Remove, Purge, Update, Upgrade :
`$ > apt-get install --name--`
`$ > apt-get remove --name--`
`$ > sudo apt-get update`
`$ > sudo apt-get upgrade`
`$ > apt-get purge --name--`
(purge removes the config as well)
or use a package manager like *'synaptic'* or *'gdebi'* like a normie.

Adding Repos to Sources.list file :
`$ > mousepad /etc/apt/sources.list` -> will open the list
Categories : main (OSS), universe (community maintained OSS), multiverse (SW restricted by copyright), restricted (proprietary device drivers), backports (packages from later releases)
Format : "deb http:// ------ --package_name-- main non-free contrib" etc.

11) FILE DIRECTORIES & PERMISSION

r,w,x -> read, write and execute

Granting ownership :
`$ > chown _username_ _file-path-name_` -> Provides the ownership of that file to that user.
`$ > chgrp _groupname_ _package-or-module-name_` -> Provides a user-group access to that module.

Checking Permissions :
3 Methods
A) `$ > ls -l _file-or-path-to-it_` -> will lay down the whole sheet. The type, permission on the file for owner/groups/users, number of links, the owner, size in bytes, creation/mod date & its name.
Eg: "drwxr-xr-x" vs "-rw-r--r--". First letter denotes directory if 'd' or file if empty dash. Followed by the permission values for 3 groups i.e owner then group then other_users. Hence we **observe 3 values at a time**. Dash means no permission ofc.
In '-rw-r--r--' -> File, owner has read/write, group and other users only have read permissions.

B) There is a proper calculation done in Octal terms as well.

```
001 : 1 : --x
010 : 2 : -w-
011 : 3 : -wx
100 : 4 : r--
101 : 5 : r-x
110 : 6 : rw-
111 : 7 : rwx
```

Total RWX is 7. Since we have 3 sets of permissions, giving a full read+write+execute permission to everyone, for example, would look like ->
`$ > chmod 777 hashcat.hcstat`

C) UGO Syntax
Here, '-' removes a permission, '+' adds and '=' sets a permission.
Eg: Remove the write (w) permission from user on a file
`$ > chmod u-w hashcat.hcstat` -> Now -rw-r--r-- becomes -r-xr-xr--
Or for user and other users at once
`$ > chmod u+x, o+x hashcat.hcstat`
Now, you can set execute permission for yourself on a newly downloaded tool/script bcz by default Linux won't set it
`$ > chmod 766 some_new_tool` -> grants us (the owner) all permission including execute -- and everyone else only R/W permissions.

Masking can be done :
`$ > umask 007` -> set it so only the user and members of the user's group have permissions.

Special Permissions :
SUID - set user ID & SGID - set group ID
1) Granting Temp Root w/ SUID
`/etc/shadow` contains all user's password -- requires root privileges to execute. SUID requires an additional bit before the permission bit. So 644 becomes 4644 i.e
`$ > chmod 4644 _file_name_`

2) Granting the Root user's Group permissions SGID

SGID works differently. Someone without execute permission can execute a file if the owner belongs to the group that has the permission to execute that file. When the bit is set on a directory -- t**he ownership of new files created in that directory goes to the directory's creator's group rather than the file creator's group.**

`$ > chmod 2644 _file_name_`

[SGID bit is represented by 2 and SUID uses 4]

Privilege Escalation :

One way is by exploiting the **SUID Bit** in the system. Eg: Scripts that need to change the password usually come with the SUID bit set already. Use that to gain temporary root priv - then do something shady like getting the file at `/etc/shadow.`

To proceed ->

Use commands like 'find' to find the files and see their bit. Example :

`$ > find / -user root -perm -4000`

Kali now starts at the top of the filesystem (because of '/') and looks everywhere below this -- the file that are owned by 'root' & specified with 'user root' + have the SUID bit set (-perm -4000).

The above command will give an output like ->

`/usr/bin/chsh ; /usr/bin/gpasswd; /usr/bin/pkexec; /usr/bin/sudo; /usr/bin/passwd,.. etc.`

Navigating to this directory, and observing, let's say "sudo", then using ls-alh, you will see ->

`-rwsr-xr-x root root 140944 _date_ sudo`

Here, the 's' in place of 'x' determines the SUID bit. Logically, anyone who runs the *sudo* file has the priv of a root user -- which becomes an attack vector IF an application -- which needs access to /etc/shadow file to successfully complete their task -- can be hijacked.

12) PROCESS MANAGEMENT

To view - use ->

`$ > ps`

Every process ofc has a PID or process ID.

You can use ->

`$ > kill _PID_value_` to kill any process.

Issue ? The *'ps'* command won't give you much info either ways. We have another command for that ->

`$ > ps aux`

It shows the USER, PID, %CPU, VSZ, RSS, TTY, STAT, START, TIME & COMMAND.

Filtering by Process Name

For instance, try running *msfconsole* command to have its process running. Then use *grep* to filter it. This way you can filter all the processes running/attached to it.

```
$ > ps aux | grep msfconsole
```

You might see a few, such as the attached DB running, the ruby script, etc, and finally the program itself.

We also have commands like **"*top*"** to monitor the processes sorted by their resource usage. It's active i.e refreshes on its own (every 3-4 seconds)

Managing Processes

We can alter the affinity/priority of any process by using the *"nice"* command (by passing a numeric value to its argument '-n'). Kernel always has the final say, we're just suggesting. The value ranges from -20 to +19.

Sadly, **the higher the +ve value, the lower is the priority and vice versa**. So -20 is most likely to receive priority, 0 is default ofc, and +19 is least likely. Usually, any process inherits the *nice* value of its parent process.

Unsurprisingly, you can alter the priority by using the *"renice"* command.

THERE IS A DIFFERENCE !!

nice is relative. Its adds/subtracts the priority value given what you pass to it. A process with a priority of 15, when asked 'nicely' to be -10 will have a priority of 5 now. OR when asked to be +5, it will now be 20. 'nice' can use the process via its location as well.

renice is absolute. Requires a fixed value b/w -20 and +19. BUT it sets the process to that level, cuz you've altered the deal and it prays you don't alter it any further. It also requires the PID.

Examples :

`$ > nice -n 10 /bin/some_slow_process` [lowers it]

or

`$ > nice -n -9 /bin/some_slow_process` [improves it]

and

```
$ > renice 20 6996
```

[6996 is the PID of some_slow_process, and 20 is setting it]

NOTE: 'top' can also be used to alter these values.

Killing Processes

'kill' command is your friend. Just pass the PID and pass the required kill signal. There are 64 of them. Default is SIGTERM (n=15) i.e termination. Ofc they are optional. Use them as a flag arg while using kill command.

$ > kill -n *PID*

Example :

```
$ > kill -9 6887
```

Signal Interrupts for kill ->
`SIGHUP (1) : Hangup - stops and then restarts with the same PID`
`SIGINT (2) : Interrupt - weak kill signal not guaranteed to work but does work mostly.`
`SIGQUIT (3) : Quit/Core dump - terminates but saves the process info in memory + inside pwd.`
`SIGKILL (9) : absolute kill signal. Forces the process to stop by sending the process's resources to a special device -- /dev/null`

Basically : to restart a process : use '-1' ; for zombie/malicious : use '-9'

Running Processes in the Background
Everything runs from within the shell and the shell waits for the task/command to run/finish. It waits for this whole sequence -- hence busy & won't allow any new commands. To prevent this we can essentially detach the process from the shell. Use the ***'&'*** right after the task.
`$ > mousepad _someDoc_ &`

Moving to foreground
Use the *'fg'* command followed by the PID. Fetch the PID if needed.
`$ > fg 1273`

Process Scheduling
Either use *'at'* or *'crond'*.
'at' is useful for scheduling a job to run once at some point in the future -- execute 1 or many commands in the future, passed with time as argument.
Eg: `$ > msfconsole at 7:20PM June 13`
OR `$ > msfconsole at now + 20 minutes`
// If you just write 'at' followed by a time, then the " **at>** " console will open asking you to map the file/process path for it.

'crond' is best for scheduling tasks to occur everyday/week/month etc. [SEPARATE CHAPTER LATER]

13) MANAGING USER ENVIRONMENT VARIABLE

Always 2 there are. **Environment** and **Shell** variables.
EnV - always uppercase, system wide, controls the way the system acts/looks/feels + inherited by child shells or processes.
ShV - usually lowercase + valid only in the shell they are set in.
Format ->
KEY=value1
OR
KEY=value1:value2:value3...

To see the Env-V's, use
`$ > env`

View All
To see vars of all types (including shell, local + shell functions) use "set" (and preferably filter it with 'more' to have a scroll-able feed)
`$ > set | more`
OUTPUT :
`BASH=/bin/bash`
`BASHOPTS=check.....:.....`
`BASH_ALIASES=()`
`BASH_ARGC=()`
....etc

Grep can be used to filter
`$ > set | grep HISTSIZE`
HISTSIZE is one such var that contains the maximum number of commands your command history file will store. It does not store the commands themselves just the number of them that can be stored.

Modify these vars
Simply set them like usual. Example:
`$ > HISTSIZE=0`

Making Var Changes Permanent
These modifications are not permanent, when done in the terminal this way. We need to *'export'* those values from the shell to the system. Since these vars are just strings, you can simply backup the contents in a text file before using the 'export' command.

For a universal backup, use the *'set'* command ->
`$ > set> ~/valueOfAllVarsToday.txt`

For singled out variables ->
`$ > echo $HISTSIZE> ~/valueofHISTSIZE.txt`
then set the new value (histsize default 1000)
`$ > HISTSIZE=0`
then export globally
`$ > export HISTSIZE`

Changing the Shell Prompt
Might seem like a cool trick but useful if you have multiple shells on different machines. That var is "**PS1**". Has 3 flags -> \u for current user's name, \h for hostname and \w for pwd name.
Again, same logic. You can use the **`echo $PS1`** command to fetch the current value

and save it somewhere before modifying how your terminal looks. Same logic to modify this var as well and then `export PS1` it to be seen globally.

Changing your PATH
PATH variable guides the system to look for commands like cd,ls, echo etc. The directories that has these commands (or even the new ones) must be added to your path else they'll be shown 'not found' even though you have them. Default stuff is mostly in `/usr/local/sbin` & `/usr/local/bin`. Each location is separated by ' : ' respectively.
Pasting my output ->
`/home/paradoxical/.local/bin:/usr/local/sbin:/usr/sbin:/sbin:/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games:/home/paradoxical/.dotnet/tools`

To add to your PATH variable :
Say a new tool is somewhere on your drive, which is not /bin and /sbin, simply append to the variable itself ->
`$ > PATH=$PATH:/home/kali/Documents/newHackingTool`
You can ofc verify by seeing the PATH value and trying to run the command directly on the terminal.
NOTE : Don't add too many directories in this var, it can hog the resources if it needs to locate a lot of directories for one command.
CAUTION : Make sure to APPEND and not OVERWRITE the variable.
Using `'PATH=$PATH:....'` and **not** `'PATH=.....'` is super important.
Something like ' $ > **PATH=**/home/kali/Docs/newHackingTool' will simply overwrite all the contents in this var. Best to save its contents somewhere.

Creating a User-Defined Variable
Simply just name and then value. Test via 'echo' command. Use it in scripts if you want after running 'export' on it. Use 'unset' to empty it.
```
$ > MYVAR="Watch_Dogs"
$ > export MYVAR
$ > echo $MYVAR
$ > unset MYVAR
```

14) BASH SCRIPTING

Very important. Apart from knowing Python/Perl/Ruby.
We run the commands directly here. As in -- more connected to the system than a GUI would let you.

Before that, some more info
`/dev/null` -> is a black-hole for info in Linux. Dump anything unwanted in it. Good for hiding the on-screen output of any tool/script

`2>` -> is basically redirecting the error from the preceding command/action. This is the 'stderr' stream. Combining this and dev/null -> `$ > _something_ 2> /dev/null` can effectively suppress the errors and send them into the black hole.

Humble Beginnings :
Start with 'shebang' which declares the shell. Comments with '#' and rest are normal commands that you write on the terminal which can be written here.

```
#! /bin/bash
# this is my first bash script
echo "Hello Friend"
```

After making such scripts and saving them, you need to set the execute permission on them.
Do that by either using `$> chmod +x` OR `$> chmod 755` before the script name.
[Difference? chmod +x will just add 'x' bit. chmod 755 will set it specifically for all groups. Meaning, if a group did not have any permission, they won't get it either way if we use '+x' but 755 will overwrite their current permission.]
And to run/execute it, simply `$ > ./script_name`

For more functionality, 'echo' can be used like `cout<<` for outputs and 'read' can be used like `cin>>` for inputs. You can directly write the var name for inputs but to use it, you need the '$' sign. Example :

```
#! /bin/bash
echo "What's your Name?"
read name
echo "Hey" $name "!. Hope you're doing well..."
```

Simple Scanner
First we'll make a basic script that uses 'nmap' with a simple TCP scan mode to scan for ports **3306** i.e MySQL services. We send the on-screen output to /dev/null and then outputting the scan results to a file using the -o flag but with a 'G' to make it grep-able.

```
#! /bin/bash
# This script is designed to find hosts with MySQL installed on local
network for testing
nmap -sT 192.168.1.0/24 -p 3306 >/dev/null -oG MySQLscan
cat MySQLscan | grep open > MySQLscan2
cat MySQLscan2
```

Self-explanatory code. To improve it, we can use variables to input the first IP Addr, the octet value for range and optionally the port.

Slightly Advanced Scanner

```
#! /bin/bash
echo "Enter the first IP : "
read FirstIP
```

```
echo "Enter the last IP (of the octet) : "
read LastIP
echo "Enter the port (3306 for MySQL) : "
read Port
nmap -sT $FirstIP-$LastIP -p $Port >/dev/null -oG MySQLscan
cat MySQLscan | grep open > MySQLscan2
cat MySQLscan2
```

[Also self explanatory code]

Bash Helpful Commands :

Command	Meaning
`:`	Returns 0 or true
`.`	executes a shell script
`bg`	puts the job in background
`break`	exits the current loop
`cd`	change directory
`continue`	resume the current loop
`echo`	displays the command arg
`eval`	evaluate the following expression
`exec`	execute the command without creating a new process
`exit`	quits the shell
`export`	Export a var/fn for all -- globally
`fg`	brings a job to foreground
`getopts`	parses args to the shell script
`jobs`	list jobs in running in bg
`pwd`	pwd
`read`	std-input
`readonly`	var declaration as read-only
`set`	list all vars (En-V & PATH)
`shift`	moves the parameters to the left
`test`	evaluate the args
`[`	conditional test
`times`	prints the user & system times
`trap`	traps a signal

Command	Meaning
`type`	shows how each arg would be interpreted as a command
`umask`	changes the default permission for a new file
`unset`	deletes a value from a var or Fn
`wait`	waits for a bg process to complete

15) COMPRESSING & ARCHIVING

'tar' is your buddy in most cases. ***'zip'*** is also a tool.
tar is "**T**ape **AR**chive" -- classic. Output is called an *'archive'*, *'tar file'* or *'tarball'*

Common args for tar, to see the contents of the archive is **'-tvf '** while compressing is **'-cvf '** & while decompressing is **'-xvf '**
-cvf = create, verbose, file to be written to
-xvf = extract, verbose, file to be extracted
-tvf = content list, verbose, file to be seen
Note : The 'v' is optional. Prefer '-tf ', '-cf ' and '-xf '. It's use case is to print on the terminal the files on which the operations is to be/being performed.
Example :

```
tar -cf my_arc.tar file1 file2 file3
tar -tf my_arc.tar
tar -xf my_arc.tar
```

Compressing Files
To actually compress a file i.e to save size, you need to use something more than tar alone.
Tar is basically lossless -- important but not space saver
3 Major commands which use 3 different algos ->
`bzip2` - uses `*.tar.bz2` -- slowest but smallest size
`gzip` - uses `*.tar.gz` or `*.tgz` -- in the middle
`compress` - uses `*.tar.z` -- fastest but biggest file size

A) gzip
GNU zip. Most common. `gzip` to compress and `gunzip` to decompress. Not only for `.tar` files but also works for simple `.zip` files.
`$ > gzip file-name.*` (applies to any file that begins with this name with any file extension)

```
$ > gunzip file-name.*
```

B) bzip2

Works in the same manner as `gzip` but has better compression ratios -- hence smaller file sizes. Very same - bzip2 to compress and bunzip2 to decompress

```
$ > bzip2 file-name.*
$ > bunzip2 file-name.*
```

C) Compress

Nothing impressive. Although, you can use gunzip with files that were compressed this way.

```
$ > compress file-name.*
$ > uncompress file-name.*
```

Bit-by-Bit Copies OR Physical Copies of Storage Devices

The **'dd'** command is the solution. Complete copy of the storage device. All you need is the path. It even copies the deleted files (because they aren't actually gone are they..)

Syntax : `$ > dd if=inputFile of=outputFile`

Example cases ->

Copying an entire 8 gig USB (usually mounted as **sdb**)

```
$ > dd if=/dev/sdb of=/root/flashCopy
```

Output :

```
1257441=0 records in
1257440+0 records out
76843809280 bytes (7.6GB) copied, 1220.73s, 5.2 MB/s
```

This command has 2 important arguments -> **`noerror`** and **`bs`**

noerror - continues the copy even if errors are encountered

bs - is blocksize. Default is 512 bytes. Typically set it to the sector size of the device (4096 bytes in our case of a USB drive). Now we can write :

```
$ > dd if=/dev/sdb of=/root/flashcopy bs=4096 conv:noerror
```

16) FILE SYSTEM & STORAGE DEVICE MANAGEMENT

`/dev` directory is for all attached devices. That includes media/hardware, (disks) physical ports, virtual host and even VGA adapter. This folder basically keeps a file for all sorts of hardware device that you may or may not use. Focus is usually `sda1/sda2/sda3` for HDDs and `sdb/sdb1` for USB devices. [In a laptop with only a NVME SSD, sda won't be there, instead we'll see 'nvme0n1p1/2/3' etc. USB drives however will surely take 'sdb' whenever they do attach.]

Newer SATA HDDs (or even ISCSI) are `sda`. The 'a' denotes the first drive. 2 **HDDs** would be `sda` & `sdb` ; which might make the **USB** connected as `sdc`. HDDs get the lettering priority before USB. That's how Linux names them. Their respective internal partitions would subsequently be sda1, sda1, sdb1, sdb2, etc.

Back in the day, Floppy Disks were mounted as `fd0` and IDE/E-IDE Hard Drives as `hd0`.

View Them

Use `$ > fdisk -l`

It lists all the partitions.

It will show the proper type i.e HPFS, NTFS, exFAT, etc which are not Linux native file systems. Linux uses ext2, ext3 and ext4.

Character & Block Devices

/dev directory will have device files whose naming will start with 'c' or 'b' - meaning Character or Block devices.

Character devices are external i.e ones that interact with the system by sending and receiving data char-by-char such as Mice or Keyboard.

Block devices are the ones which communicate in blocks of data & include devices like HDD and DVDs. They require higher-speed data throughput & therefore send/receive data in

in blocks. Our major focus is Block devices for further commands

List Block Devices Info

Use `$ > lsblk`

Very similar to `fdisk -l` but it also displays devices with multiple partitions in a kind of tree + showing each device with partitions as branches + does not require root priv.

This will also list the mount points (denoted by '/').

My output :

```
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINTS
nvme0n1 259:0 0 238.5G 0 disk
├─nvme0n1p1 259:1 0 25G 0 part /media/paradoxical/DataDrive
├─nvme0n1p2 259:2 0 94.2G 0 part
├─nvme0n1p3 259:3 0 977M 0 part /boot/efi
├─nvme0n1p4 259:4 0 112.1G 0 part /
└─nvme0n1p5 259:5 0 6.2G 0 part [SWAP]
```

Mounting and Unmounting

The term itself comes from the days of storage tapes. Every device needs a mount point i.e some directory to be mounted to. 2 main are `/mnt` and `/media` altho you can use any directory. External devices pick /media and internal ones are for /mnt.

Make sure the directory you mount to (/mnt or /media) is empty or else after mounting the sub-directories will get over-written. And as always, all the file-systems info is stored in **/etc/fstab.**

Examples ->

```
$ > mount /dev/sdb1/ /mnt
$ > mount /dev/sdc1/ /media
$ > umount /dev/sdb1
```

Monitoring Filesystems

For simple stuff, use 'disk free'

`$ > df` -> gives us basic info on any HDD/Mounted device like CD/DVD/Flash Drives. Without any args, it fetches the first drive of the systems (usually `sda` but now it might be `nvme0n1p1`). For a specific drive, use `$ > df sdb`.

My output for reference :

```
Filesystem 1K-blocks Used Available Use% Mounted on
udev 7967092 0 7967092 0% /dev
tmpfs 1615372 1668 1613704 1% /run
/dev/nvme0n1p4 115142704 49942804 59304732 46% /
tmpfs 8076856 16136 8060720 1% /dev/shm
efivarfs 184 121 59 68% /sys/firmware/efi/efivars
tmpfs 1024 0 1024 0% /run/credentials/systemd-journald.service
/dev/nvme0n1p1 26214396 6908512 19305884 27%
/media/paradoxical/DataDrive
/dev/nvme0n1p2 98813948 69000 98744948 1% /media/paradoxical/MoreData
tmpfs 8076856 8 8076848 1% /tmp
/dev/nvme0n1p3 998480 27076 971404 3% /boot/efi
tmpfs 1024 0 1024 0% /run/credentials/getty@tty1.service
tmpfs 1615368 120 1615248 1% /run/user/1000
```

Checking for Errors

File-system check or 'fsck', will check for errors, repair the damage if possible or else puts the bad area into a bad-blocks table to mark it as bad. For this command, you need to specify the file-system type of the device file to check (default is ext2).

NOTE : For **fsck** to work, you NEED to unmount that drive first (by running `$ > umount _device-path_`)

```
$ > fsck -p /dev/sdb1
```

Here, the '-p' flag will automatically repair any problems with the device.

17) LOGGING SYSTEM

Log files are very important. For all sorts of users. All activity trail is logged, irrespective of the user doing that activity. As a grey hat, your purpose is to hide/destroy/scramble the evidence.

First we focus on the logging service daemon.

Logging Daemon

Actual daemon was **`syslogd`** but it has 2 variations, `rsyslog` and `syslog-ng`.

Debian uses rsyslog so that's our focus.

Use `$ > locate rsyslog` to find files on it.

Output would be huge but what concerns us is :

```
/etc/rsyslog.conf
/etc/rsyslog.d
```

```
-------
/etc/logrotate.d/rsyslog
/etc/logrotate.conf
```

rsyslog Config File
Managed + configured by a plaintext file, located in the `/etc` directory, and for us it is `/etc/rsyslog.conf`.
Opening it in mousepad will reveal us a lot of modules for various logging purposes. (imuxsock, imklog, immark, UDP, TCP etc). Focus is around line 50 i.e the **Rules Section**.

rsyslog Logging Rules
Determines what kind of information is logged, what programs have their messages logged and where is that log stored. Line 50 in the config file reveals :
Format = `facility.priority action`

```
# Log anything besides private authentication messages to a single log file
*.*;auth,authpriv.none -/var/log/syslog
# Log commonly used facilities to their own log file
auth,authpriv.* /var/log/auth.log
cron.* -/var/log/cron.log
kern.* -/var/log/kern.log
mail.* -/var/log/mail.log
user.* -/var/log/user.log
# Emergencies are sent to everybody logged in.
*.emerg :omusrmsg:*
```

[In my output above, statements like daemon -/var/log/daemon.log or mail logging info section was missing.]

`facility` refers to the program whose messages are being logged, `priority` determines what kind of messages are being logged and `action` references the location of where the log will be sent.

`facility` is :
`auth` / `authpriv` - Sec/auth messages
`cron` - clock daemons
`daemon` - other daemon
`kern` - kernel messages
`lpr` - printing system
`mail` - mail system
`user` - generic user-level messages
`*` - means all facilities

`priority` is :
`debug`, `info`, `notice, warning*, warn*, error*, err*, crit, alert, emerg*, panic`
(ones in `` are deprecated. 'debug' has lowest priority and 'panic' has highest. Messages classified as 'alert' will stay on that priority and won't drop down to any lower level).

`action` is : filename and location of where the logs should be sent. Usually it is `/var/log` directory with a filename that describes the facility generated. Eg: logs for '`auth`' facility would be sent to `/var/log.auth.log`.

Examples :
`mail.* /var/log/mail` - log mail events of all (*) priorities to /var/log/mail*
`kern.crit /var/log/kernel` - log kernel events of critical priority or higher to /var/log/kernel
`.emerg *` - log all events of the emergency priority to all logged on users.

Automatically Cleaning Up Logs with LOGROTATE
Even log file takes space. Don't delete them for too long and they take up space. Delete them too often and you won't have logs for investigation. **"logrotate"** determines the balance b/w these values. It simply means to archive the logs in a timely manner. Old files are replaced with new files. The config file is located in `/etc/logrotate.conf` text file. A cron job already handles the schedule for logrotate. Opening this file we see a few 'variables' :
`weekly` - unit of time the rotate number refers to
`rotate` - the number working on the unit specified
`create` - create new empty log files after rotating the old ones
(optionally/commented) `compress` - if you want to
`include` - the directory where the log rotate info is dropped

Sample values :
`weekly`
`rotate 4` = rotate logs every 4 weeks
`create`
`include /etc/logrotate.d`

At the end of each rotation period, log files are renamed and pushed toward the end of the chain of logs as a new log file is created, replacing the current log file.
Eg: `/var/auth.log` becomes `/var/auth.log1` then `/var/log/auth.2`, so on up until auth.4, post which instead of auth.5 won't be created as the last one is rotated and a new one is created.
Try `$ > locate /var/log/auth.log` and notice only 4 files are visible.

Remaining Stealthy

2 ways - Removing Evidence and Disable logging itself.

A) Removing Evidence

You can ofc use 'rm' and delete all the files, but there's a better method, plus there can be gaps in the log files which might look suspicious. Instead use -

```
$ > shred -f -n 10 /var/log/auth.log.*
```

This command basically scrambles the data by re-writing inside it many times. Simply deleting them, would cause them to be recovered by some pro person. Simply scramble them and delete them.

Here, `-f` flag stands for changing the file permission to allow over-writing. `-n` stands for the over-writing count. In our example, the file is being overwritten 10 times. The '`*`' symbol will find all occurrences of the file + find the ones made by 'logrotate' as well.

B) Disabling Logging

Requires root privileges. Disable it just like any other services' daemon.

```
$ > service rsyslog stop
```

(stop, start, restart are the args)

NOTE : Disabling `rsyslog` will also make a note in the log file in `var/log/syslog` - that the service was stopped

18) USING & ABUSING SERVICES

Anything running in bg is service. Many exist. Our focus is : Apache Web Server, OpenSSH, MySQL & PostgreSQL.

First, remembering how to handle services. Start, stop, restart. Example :

```
$ > service SERVICENAME start
$ > service SERVICENAME stop
$ > service SERVICENAME restart
```

Creating an HTTP Web Server w/ Apache

Typing `$ > sudo service apache2 start` will start the server.

Going to `http://localhost/` will open the default apache landing page. The HTML for this can be changed by editing the `index.html` file located at `/var/www/html`. Make sure to `sudo` the mousepad command to overwrite the html file.

OpenSSH & R-Pi Spy

Requires a R-Pi. Author used R-Pi 2 + the camera module. Will revisit this section when I'll buy a R-Pi 5. Learning about SSH is shallow here.

`$ > sudo service ssh start` is to be enabled/done inside the R-Pi.

Then from your Kali installation, use `$ > ssh pi_username@192.168....` followed by the password you set on the R-Pi. Rest is irrelevant as of yet.

Extracting Info from MySQL DB

First start the service,

`$ > sudo service mysql start`
Next, authenticate.
`$ > mysql -u root -p` -- as this was dictated by the author. It won't work. Instead, go with :
`$ > sudo mysql`
After that, most probably a `MariaDB [(none)] >` prompt will open. MariaDB is the true open source fork of MySQL, detached from Oracle. Now in this prompt, you can simply type the SQL queries like it's 2017. Also, semicolons are relevant here, don't forget.
Running `MariaDB [(none)] > show databases;` -- will print all 4 default DBs. 2 are administrative (the 'schemas'), one is 'sys' and the last is 'mysql' on which you can experiment for now.
`MariaDB [(none)] > select mysql;` -- will select it.
And the prompt will become
`MariaDB [(mysql)] >` -- now you can exfiltrate data as you want.

Optionally - you can change the user password for root on MySQL by using :
`MariaDB [(mysql)] > update user set password = PASSWORD("12345678")`
`wheere user = 'root';`

Author added a section, wherein he connects to a different DB on the local network. Can be done via `$ > mysql -u root -p 192.168.1.101` which will again prompt for a password. Then he simply connects to a DB, conveniently named 'creditcardnumbers'.
`MariaDB [(creditcardnumbers)] > show tables;`
Using that, we observe a table named 'cardnumbers'. We use `MariaDB [(mysql)] > describe cardnumbers;` to observe the table, which has field like Address, Name, City, State, etc. Then we use `MariaDB [(creditcardnumbers)] > SELECT * FROM cardnumbers;` to basically view everything all at once. Now export it how you want.

PostgreSQL with Metasploit
If not installed, then install it. Post which, start the service by :
`$ > sudo service postgresql start`
For this exercise, we need Metasploit, which cab be run via `$ > sudo msfconsole` and the console shell prompt changes to metasploit.
Objective : is to make a PostgreSQL DB on which metasploit can save the scan results and also save its modules to fetch from, increasing speed.
Inside Metasploit :
`msf > msfdb init`
Output :
`[*] exec: msfdb init`
`[i] Database already started`
`[+] Creating database user 'msf'`

```
[+] Creating databases 'msf'
[+] Creating databases 'msf_test'
[+] Creating configuration file '/usr/share/metasploit-framework/config/database.yml'
[+] Creating initial database schema
```

Now we log in to Postgres as root. Use the switch user 'su' command on the msfconsole.
`msf > su postgres`
Console now changes to :
`postgres@paradoxical:/home/paradoxical $ >`

Now here, we run a createuser command with the username 'msf_user' and the '-P' flag to setup a password :
`postgres@paradoxical:... $ > createuser msf_user -pP
(Console prompt will appear for password. 12345678 for this example)

Now after all this, Make the actual DB and grant this user the permission for the same.
`postgres@paradoxical:... $ > createdb --owner=msf_user MyCustomDB`
Then just type `exit` and you'll fallback to the msfconsole prompt.

Now just connect the Metasploit instance to this DB.
`msf > db_connect msf_user:12345678@127.0.0.1/MyCustomDB`
Output : `[*] Connected to Postgres data service: 127.0.0.1/MyCustomDB`
[Why localhost? Because we made this DB locally. The IP would change if it was a DB deployed somewhere else remotely.]

Then check status by
`msf > db_status` -- and it should show :
`[*] Connected to MyCustomDB. Connection type: postgresql. Connection name: local_db_service.`
If it's not connected, the output will be `[*] postgresql selected, no connection` instead.

19) BECOMING SECURE & ANONYMOUS

Everyone is behind your ass. Simple as that. You are the commodity if the service is free.
When you send a request the packet is tracked either ways. It contains the IP Addr of the source and destination -- cuz packet knows where its going and where it came from. Literal hopping of this packet is tracked. That's how websites know who you are when you come back.
`$ > traceroute google.com` will show you how the packet hops through many servers that google owns.

TOR

Savior. Decentralized. Works on top of all the TOR Routers established worldwide. A separate network. At each hop, the info is encrypted then decrypted by the next hop when it's received -- this way, each packet only contains info about only the previous hop and not the IP Addr of the origin & the website owner can only see the IP of the last router that sent the traffic. TOR can be slow that's the issue. Another issue? TOR has been broken by NSA so there's a good chance you're surfing over their routers + they use a method called 'traffic correlation' -- which looks for patterns to break TOR's anonymity. You are safe from corps like Google but not from NSA/FBI.

Proxy Servers

When you have a middleman b/w your connection, it becomes a proxy. The traffic adopts the IP Addr of the proxy(/ies) when you surf through them. Accountability shifted to the proxies and not the originating IP (i.e you). To make everyone else's life difficult, just use "proxy-chains" - a tool that literally configures proxies for your local machine & comes pre-installed in Kali. Just configure it once and use this tool before you run any application/tool which involves data coming from your machine such as firefox browser or a Nmap scan.

Example :

```
$ > proxychains nmap -sS -Pn IP_ADDR
```

OR

```
$ > proxychains firefox website-domain-name-here
```

To configure, visit `/etc/proxychains4.conf` [Author mentions `proxychains.conf` which is replaced with `proxychains4.conf`]. This config file has a lot of values, some active, some inactive -- all decided by what is commented and what isn't. What matters to us is :

```
[ProxyList]
# add proxy here
# meanwhile
# defaults set to tor
socks4 127.0.0.1 9050
```

Replace your proxy values here. You can buy premium ones or get some for free but remember the tradeoff. The format remains the same -> `type IP-Add port`

Example : `socks4 114.134.186.12 22020`

Make sure to comment the default localhost proxy value (handled by TOR) and write your own. Now :

```
[ProxyList]
# add proxy here
socks4 114.134.186.12 22020
# meanwhile
# defaults set to tor
# socks4 127.0.0.1 9050
```

Tip : You can add multiple proxies here, essentially making a chain. Chaining is sequential but automatic if proxychains detect multiple values here in this space.
For dynamic chaining, make sure to un-comment the `dynamic_chain` variable in the config file while comment out `strict_chain`. Dynamic is better as it auto switches if one proxy is down/not-responding.
Lastly, Random Chaining can also be done. Kinda like Dynamic Chaining but picks the proxies randomly. Simply **comment out** `dynamic_chain`, `strict_chain` & **un-comment** `random_chain` and set `chain_len` to something like `chain_len = 3`.
Trade-off ? Latency.
All depends on how good your proxies are.

VPN and Encrypted Mail
Self-explanatory. Buy the good VPN you can afford (ProtonVPN) and try to use ProtonMail. A VPN that offers proper tunneling and actively fights the bad-laws is your go-to VPN.

20) WIRELESS NETWORKS

Understanding and Inspecting them.