# Linux essentials writeup

## 0. Foreword

In order to pass my exams I kinda have to study but its just something I just don’t do.. I’m the type of person who doesn’t care about studying and just likes to do put everything into practical use and that’s what I’ll do for this writup of Linux Essentials.

In here I’ll write down all commands with screenshots, code and explanations in order for other people and myself to learn from it.

## 1. Working with GUI

### 1.0 Introduction

In here nothing really important was mentioned we mostly went over what Linux is and how to set up a virtual machine..

### What is Linux

#### 1.1.1 What Is Linux

In here its once again about what Linux is but there is some useful stuff to look at.

So obviously, Linux is an operating system like Windows, Android and MacOS.

Graphical user interface, application

Description automatically generated  
Linux comes from Unix, Unix is also an operating system. It supports multitasking and multi-user functionality. It has a graphical user interface just like Windows to support easy navigation and support.

Linux was build up through 3 factors:

* MINIX Operating system (Open source Unix like OS)
* GNU Project (Unix clone with bad kernel)
* Linus Torvalds (Some random IT student that wanted to flex)

#### 1.1.2 Linux Kernel

Linux on its own is actually only a Kernel while an Operating system like Ubuntu or Kali is a combination of Linux and extra software.

The Kernel is the core of an OS and is for example responsible for communicating with the hardware.

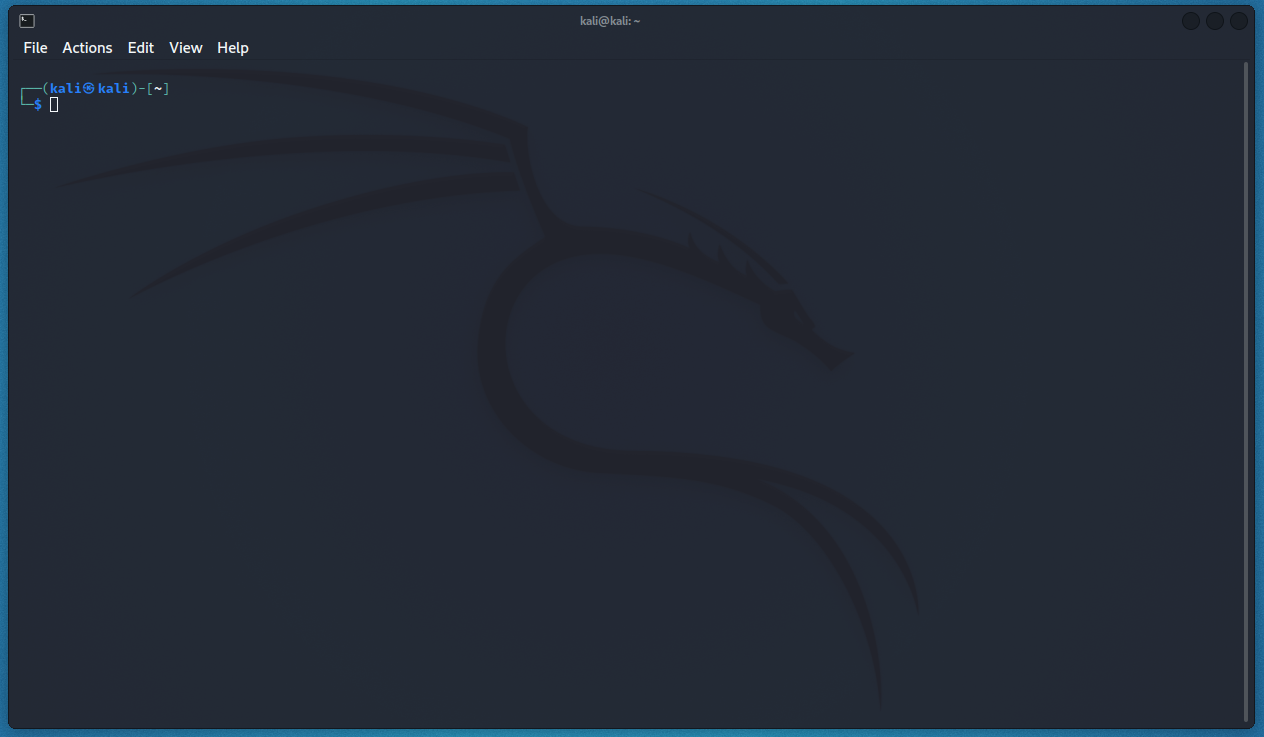
#### 1.1.3 Linux CLI

Linux CLI is a text interface, with it you have no need at all for a graphical interface. Using CLI is really powerful and its an important skill to master, OS like Arch Linux are completely build on only using the CLI and it gives its users a lot more power than you can get from a graphical interface.

Graphical user interface, diagram

Description automatically generated with medium confidence

CLI has 3 parts that should be pointed out right now.



Prompt

Shell

Terminal

Prompt: A piece of text given by the shell that tells when you can send a new command.

Shell: Shell is the text program that waits for text input that will handle all commands and programs.

Terminal: Application that shows the CLI.

We will mostly use Bash aka. Bourne Against Shell, this is also a CLI Shell but it also contains command language in which programming is possible.

## 2. Working with CLI

### 2.0 Special characters

Some characters are more difficult to find than others, on my Belgian keyboard it can be a pain in the ass so time to write those key combo’s down!

~ = ALT GR + ‘=’ + <Space>

` = ALT GR + ‘µ’ + <Space>

### 2.1 Man Pages

#### 2.1.1 What are Man Pages

We will first go over Man pages, this is short for Manual pages and in short we can use Man <command> To get an entire manual page of a command.

#### 2.1.2 Usage & Navigation

We can use Man ls to get the Man Page of the command ls.

A screenshot of a computer

Description automatically generated with medium confidence

Within the man page we can scroll but there are multiple other ways to navigate through the pages!

Within the man page we can use H to find all commands within the man page.

We can use the following things to navigate:

* **Space or f**: Move 1 page down
* d: Move down half a page
* d: Go back one page
* d: Go back half a page
* /: Jumps to the first word you type before you press enter after you press enter you can use
  + space or n to jump to the next
  + shift + n to move back
  + \btext\b to search for a specific word (when just searching for text it will also match with context, this wont)
* g: go to the first line of the man page
* Shift + g: go to the last line of the man page
* q: Quit the Man Page

#### 2.1.3 More info

Most Unix files have a man page, we will go over some with examples:

* Man <Command>: Man following a command will give out the manual with all variations | man ls
* Man <Config-File>: Most config files have its own manual page | man resolv.conf
* Man <daemon/root-binary>: man pages also exist for daemons | man system-networkd

We can also use man -k <string> to search through all Man Pages with the given string inside.

We can use whatis <path> to get a description | whatis route

We can use Whereis <path> to get the location of a Man Page | whereis passwd

There is also Man Man which will give the man page of man.  
In here we can see:

* 1 Executable programs or shell commands
* 2 System calls (functions provided by the kernel)
* 3 Library calls (functions within program libraries)
* 4 Special files (usually found in /dev)
* 5 File formats and conventions eg /etc/passwd
* 6 Games
* 7 Miscellaneous (including macro packages and conventions), e.g. man(7)
* 8 System administration commands (usually only for root)
* 9 Kernel routines [Non standard]

These are different sections within a man page.

We can search for a section in a man page by using man <sectionNr> <path> | man 5 passwd

### 2.2 Working With Directories

#### 2.2.1 What are Directories

In short: A Directory is a location for storing files on your computer. These sit in a hierarchical file system.

#### 2.2.2 Navigating through Directories

We can use pwd (Print Working Directory) to check the current directory you are in from the root (/) .

We use cd (Change Directory) to change our directory, when we use ls we can see to what Directory we can quickly change to |   
cd Downloads.

Cd has some special ways to path:

* ~ : Instantly go to the home directory | cd ~/Downloads
* .. : Move one folder up | cd ../Downloads
* . : Stay in the current directory | cd .
* - : Go to the directory you where previously in | cd -

Absolute and relative paths, to put it really simple, an absolute path always starts with ‘/’ in Linux, a relative never starts with ‘/’.

When looking for downloads from for example the home folder you will do:

* Absolute path: cd /home/<name>/Downloads
* Relative path: cd <name>/downloads

An Absolute path always starts from the root while a relative starts from the directory you are currently at.

We can see the current directories and files using ls inside of a directory.

Ls has some important arguments to remember:

* -a : This will show all files including hidden files | ls -a
* -1 : when adding -1 your folders will be listed vertically | ls -1
* -l : When adding -h the file sizes will be shown we add 1 to it to make the listing more visible | ls -l

#### 2.2.3 Tree

Tree is a package that isn’t on Linux by default, we can install this using sudo apt install tree more info about this later.

Now we can use the tree command and receive a directory tree, we can find in the man page many different parameters we can add to it using man tree.

In this example I made a tree of everything inside the home folder.

Text

Description automatically generated

#### 2.2.4 Create and remove directories

Obviously we can also create and remove directories

We can simply do mkdir <string> to create a directory with name | mkdir hello.

We can also use mkdir -p <string> to create a directory within parent directories, when adding -p mkdir will create the directories that don’t exist yet | mkdir -p ~/hello/hello/hello.

Text

Description automatically generated

You can delete empty directories using rmdir <path> |   
rmdir ~/hello/hello/hello

Text

Description automatically generated

You can also delete an entire path with rmdir -p <path> |   
rmdir -p hello/hello

Text

Description automatically generated

You can **not** add a ~ to your path cause with you tell rmdir to also remove your home folder like that!

### 2.3 Working With Files

#### 2.3.1 Files In Linux

Files in Linux are case sensitive do if you search for File1 by with the command cat file1 you won’t find it!

Its also important to know that basically everything on your OS is a file, even directories are some kind of special file and is still case sensitive!

If we want to see what type of file extension a file is we can use the file command | file /etc/passwd.

#### 2.3.2 File create, copy, remove and move

If we want to create a file e can simply do touch <string> in order to create one | touch hello.

You should keep in mind that touch can do this:

* -t : When you want to specify a time created instead of the current time | touch -t 201905050000 hi

If we want to remove a file e can simply do rm <path> in order to remove one | rm hello.

Rm has some interesting parameters to look at:

* -i : When adding -i to rm you will get asked for a yes or no if you really want to remove the file | rm -i hello
* .\* : Removes all hidden files \* stands for everything so this basically removes everything with a . in front of it | rm .\*
* {\*,.\*}: Removes all files and hidden files | rm {\*,.\*}
* -rf : This is a yeet away all, rm -r will not remove non-empty directories but when we add the -f parameter it will force it to also remove non-empty directories | rm -rf hello

In order to copy a file we can use the cp command | cp hi hi2:

* We can also copy a file into a directory with the same name as the target file | cp hi hiDirectory/
* -r : You can also copy an entire directory with -r |   
  cp -r dir1 dir2
* -i : when we add -i in front we can prevent cp from overriding any existing files | cp dir1 dir1BackupWhereFilesCantChange

We can Ofcorse also move and rename files for this we use the mv command with this we can chose a file and place it in a directory by doing mv <targetFile> <Directory> | mv file1 dir2/

What this does is delete file1 at 1 spot and placing it in another we can also use this to rename and simply say mv <targetFile> <newname> what this does is it will recreate the file at the current path, which is the same with a new name | mv file1 file2

There is also the rename command but it’s a little more tricky. It uses regular expressions but we will get into it. Imagine that we got a folder with .txt files and we want them to be .png files in order to do this we can do | rename ‘s/\.txt/.png/’ \*.txt

* First it does the rename command
* Then you specify .txt which is the thing you want to be changed | ‘s/\.txt
* Then you specify what you want to replace .txt with .png | /.png/’
* This happened between quotes to make sure its seen as one string | ‘s/\.txt/.png/’
* Then we specify our targets which is all files with .txt in the back | \*.txt

### 2.4 Working with file contents

To display the first 10 lines of a file we can use the head command | head /etc/passwd .

If we wanted to for example only see the first 4 we can do head -4 this will only display the first 4 lines | head -4 /etc/passwd.

Tail is the opposite of head, it will display the last 10 lines of a file or the specified amount | tail -4 /etc/passwd.

Then we have cat, this is one of the most universal tools but this actually only copies the standard input to standard output, this will output all the contents | cat /etc/resolv.conf.

Cat stands for concatenate and then we can obviously concatenate files together, here is an example:

I’ll first create 3 files with text I’ll use echo and add it to an output stream, more details about that later:

Graphical user interface, text

Description automatically generated

Then we can see the content of those files:

Text

Description automatically generated

Now we want to put all those files into one we can once again use the output stream and do:

Text

Description automatically generated

We can use cat also to create a new file we do this by simply doing cat > <filename.extension> we will see more about what ‘>’ means in 2.12 I/O redirection | cat > hi.

When doing this we will need to type your text after doing enter on cat > hi and use ctrl + d to tell the cmd that it’s the end of file.

We don’t have to use ctrl + d tho, we can specify a stop commando using << when de do cat << <stop-command > hi.txt we can specify a stop commando and when we then write the command cat will stop | cat << stop > hi.

Text

Description automatically generated

You can also copy files with cat by simply taking the input with cat <input> and use the > to put it in another file |   
cat input > copyFile.

There is cat and tac, cat reads from up to down and tac reads from down to up | tac file.

With strings you can get readable ascii strings found in files like binary files | string /bin/ls.

### 2.5 File System directory structure

In here we will go over the most common directories in the Linux file tree

Really important is to know and understand the man hier command this will explain the directory structure hierarchy of your Linux distribution

Here are some important directories:

* Root : this is represented with a / everything that exists on your Linux system is here | ls /
* /boot : this directory contains the files needed to boot the computer | ls /boot
* /bin : in here there are binaries these are files that contain compiled source code, these are sometimes called executables | ls /bin
* /sbin : this contains binaries to configure the operating system (these require boot privileges) | ls /sbin
* /etc : in here its machine specific config files | ls /etc
* /etc/skel : this is copied to the home directory of a new user |   
  ls -A /etc/skel/
* /etc/sysconfig : contains a lot of Red Hat Enterprise Linux configuration files | ls /etc/sysconfig
* /home : all the users are stored here | ls /home
* /root : default location for personal data and profile |   
  sudo ls -A /root
* /srv : Data server by your system | ls /srv
* /lib : /bin and /sbin use shared libraries from /lib | ls lib
* /media : used for removable media devices | ls /media
* /mnt : this is a temporary mount point and is mostly used for remote file systems | ls /mnt
* /opt : store optional software, this mostly comes from outside the distribution repo | ls /opt
* /tmp : used to store temporary data
* /dev : files that are not located on the hard disk but its for the kernel to recognise hardware devices | ls -d /dev/[stp]??
* /dev/null : basically a black hole, it has unlimited space but everything you put in you can’t get out | echo hello > /dev/null
* /proc : this is used to communicate with the kernel |   
  cat /proc/meminfo
* /sys : contains kernel info | ls /sys
* /usr : this should only contain shareable, read only data | ls /usr
* /var/www : website data is saved here | ls /var/www
* /var/log : centerpoint for all log files | ls /var/log
* /var/log/syslog : first file to check when troubleshooting on Debian, this contains info what just happened on your system |  
  var/log/syslog
* /var/log/messages : used to check when troubleshooting on Red Hat, this also contains information on what just happened to the system | sudo ls /var/log/messages

### 2.6 Commands and Arguments

Now we will go over shell expansion and take a close look at commands and arguments

One of the primary features of a shell is to perform a command line scan. When you enter a command the shell will cut all parts up and execute the command accordingly with all arguments attached.

Its important to know that parts that are separated by one or more white spaces are considered separate arguments.

We can use single quotes to prevent the removal of white spaces |  
echo ‘a line with single quotes and m ultiple white spaces!’

The same works with double quotes |

echo “a line with single quotes and m ultiple white spaces!”

We can use special characters in quotes but to do so in the echo command we need to add -e behind it |  
echo -e “A line with \na a new line” |   
echo -e “A line with \ta a tab

External or building command?

The difference that not all commands are external to the shell, some are building. External commands have their own binary mostly in /bin or /sbin, building commands are integral to shell itself.

We can use type to check if a command is external or building |  
type cd | type cat  
When we do this with ls we see that type also tells if a command is aliased | type ls

With which we can find the absolute path of commands  
| which cp ls cd mkdir pwd

We can create an Alias for commands if we wish to, for this we use the alias command | alias dog=tac

A screenshot of a computer

Description automatically generated with low confidence

You can also use an alias to set default options for example if we want -i to be behind rm by default we can add this in the alias   
| alias rm=’rm -i’

When we do just alias we can see a list of aliases | Alias

To remove an alias we can use unalias | unalias dog

### 2.7 Control operators

There are control operators in the shell we can use this to change the flow of our commands

Semicolon ; can be used to separate commands   
| echo hello ; echo world

Ampersand & when an a line ends with an ampersand the shell won’t wait for the command to finish, you’ll get your prompt back and the command is executed in the background | sleep 20 &

Dollar question mark $? The exit code of the command executed before is stored in this shell variable | echo $?

Double ampersand && this is a logical and, the command after this operator will only execute if the one before succeeded (0 exit status)  
| zecho this command is wrong && echo this won’t run bit is right

Double vertical bar || this is a logical or, this will only run if the command in front fails | echo this will run || echo this won’t run

This can be combined in longer lines, for example if you want to echo if a file is successfully removed or not  
| rm file1 && echo It worked || echo it failed

Pound sign # pound sign is used as a comment, everything behind will be ignored | echo hello world #Whatever I write here is ignored

Escaping special characters backslash \ with backslash you can use special characters in shell without them acting as control operators  
| echo ‘console.log(“hello world)\;’

Splitting command line backslash you can also use a backslash to split long lines   
| echo this is the first line \  
 This is the second line \  
 This is the third line

### 2.8 Shell variables

Shell variables are used to manage environmental variables in the shell, these are often needed by applications, its recommended to not mindlessly change these values.

Dollar sign $ when there is a $ in front of a string it will look for an environment variable named with the string | echo $SHELL

Its important to note that these variables are case sensitive

You can also create variables using = | myVar=555   
When calling for myvar it has to be between “ and not ‘   
| echo myVar | echo “myVar” | echo ‘myVar’

When using the set command you can get a list of shell functions and variables | set

To remove a variable you can use unset | unset myVar

PS1 is used to define the shell prompt DON’T JUST TRY IT WITHOUT BACKING UP YOUR /etc/bashrc when doing this you will change your console and its pretty ugly! | PS1=hello

$Path is the variable used to determine where the shell is looking for commands| Path=$Path

Env contains exported variables, when calling set you’ll receive a list of all exported variables | env

With export you can export shell variables to other shells, this will export the variable to child shells | export myVar

When combining shell variables with other string values we use ${shell variable}<string> instead of shellVar<string> for obvious reasons | echo hello ${prefix}Daan greetings

Unbound variables that come when trying to echo a variable that doesn’t exists shell will give nothing back, you can enable and disable this | set -u | set +u

### 2.9 Shell embedding and options

Shell can be embedded on the command line, this means that the command line scan can spawn new processes containing a fork of the current shell.

We can see it through this example

Text

Description automatically generated

Backticks is another version of the dollar bracket ${} embed, backticks ` are not single quotes ‘  
| echo hello there today its `date +%A`

### 2.10 Shell history

Shell makes it easy to repeat commands here we go over how.

We can use !! to repeat the last command this is pronounced as bang bang | !!

To repeat the last command that starts with a specific string we can do !<string> | !dat

We can use the command history to see all commands we used we can put a value behind it to see for example the last 100 commands

| history 100

If we want to repeat a command from the history page we can use !<command number> |!13

We can also use CTRL+R with this we can search for keywords to find out command in the history

The variable $HISTSIZE determines how much commands will be remembered | HISTSIZE=25000

The variable $HISTFILE points to the file that contains your history

$HISTFILESIZE returns how many commands that are kept in your history

If you want a command to not be recorded you can use a space in front of your command

It is possible to use regular expressions with bang we can for example create a file named file1 and copy the process and change 1 to 2  
| !c:s/1/2/

### 2.11 File globbing

The shell is also responsible for file globbing (dynamic filename generation).

Asterisk \* the \* is interpreted by the shell as a sign to generate filenames, when no path is given the shell will use filenames in the current directory, when doing ls file \* the shell will return all files with file as their first letters, this is case sensitive | ls file\*

Hidden files are by default not included but we can do so by enabling dotglob | shopt -s dotglob | shopt dotglob

Question mark ? with the question mark you can check if everything except the question marked character is equal, if we have filo4 file4 ls fil?4 will find those 2 | ls fil?4

Brackets [] are used an array of potential matches  
| ls file[245][abcdefghijklnmopqrstuvwxyz]

In between brackets we can also use ranges [a-z] and [0-9]

We can change the case sensitivity, bij default this is already off but we can disable this | shopt – globasciiranges

Even now sometimes cases aren’t case sensitive this is cause of the settings in $LANG we can change this so it actually will| LANG=C

If we have case sensitivity we can use [a-zA-Z] for example to have both upper and lower case.

If we don’t want file globbing to happen we can use a \\* or enclose the \* in “ or ‘

Ofcorse we can also combine these things! | \*[fF]ile??

### 2.12 I/O redirection

### 2.13 Filters

### 2.14 Basic Unix tools

### 2.15 Regular expressions

### 2.16 Working with vi

### 2.17 Users

### 2.18 Groups

### 2.19 Standard file permissions