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

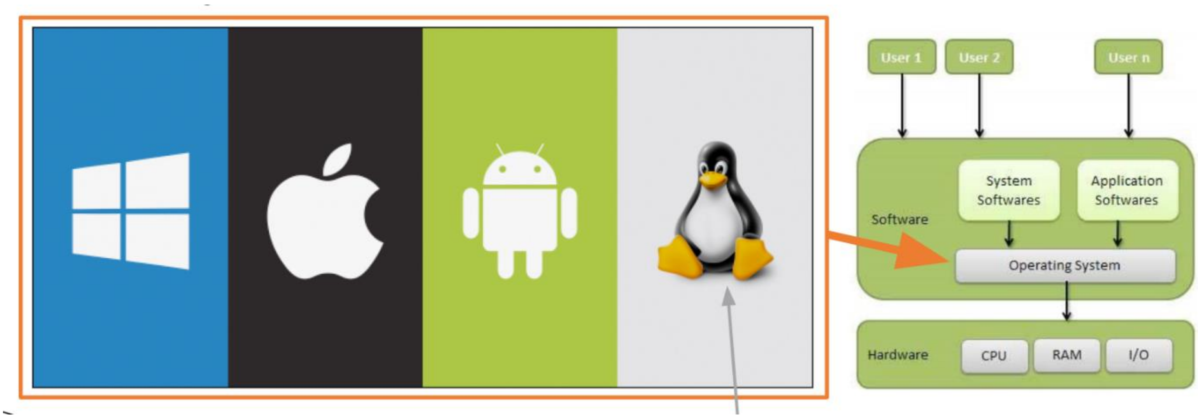
In here nothing really important was mentioned we mostly went over what Linux is and how to set up a virtual machine..

1.1 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.



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)

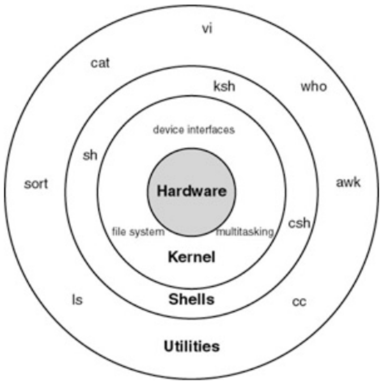
1.1.2 Linux Kernel

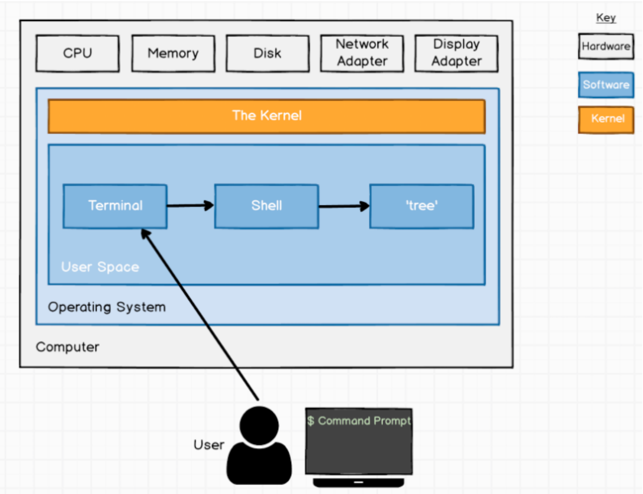
Linux on its own is actually only a Kernel while an Operating system like Ubuntu or Kali is a combination of Linux and extra software.

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

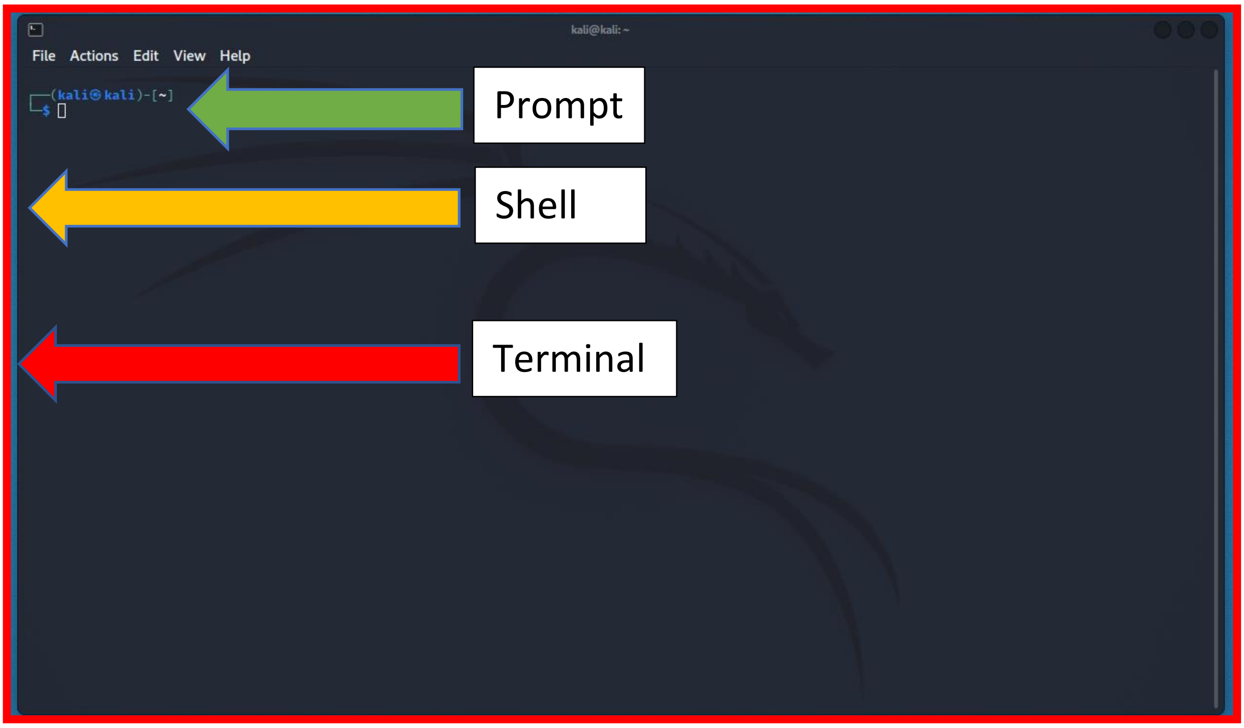
1.1.3 Linux CLI

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.





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



```
┌──(kali㉿kali)-[~]
└─$ ▯
```

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>

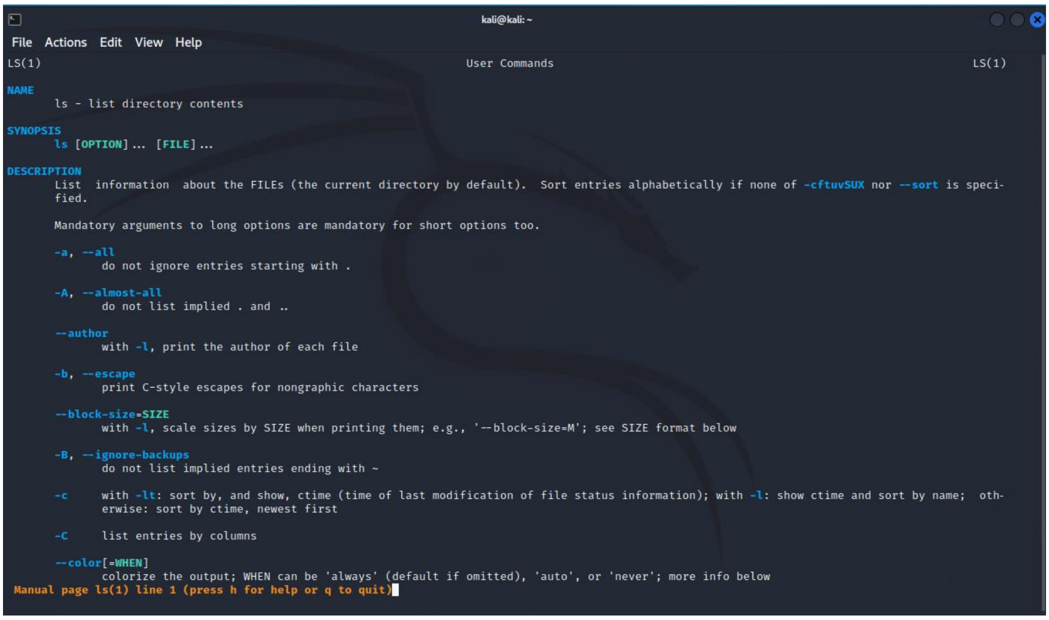
2.1 Man Pages

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.



```
LS(1)                                                    User Commands                                                   LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List  information  about the FILEs (the current directory by default).  Sort entries alphabetically if none of -cftuvSUX nor --sort is speci-
       fied.

       Mandatory arguments to long options are mandatory for short options too.

       -a, --all
              do not ignore entries starting with .

       -A, --almost-all
              do not list implied . and ..

       --author
              with -l, print the author of each file

       -b, --escape
              print C-style escapes for nongraphic characters

       --block-size=SIZE
              with -l, scale sizes by SIZE when printing them; e.g., '--block-size=M'; see SIZE format below

       -B, --ignore-backups
              do not list implied entries ending with ~

       -c     with -lt: sort by, and show, ctime (time of last modification of file status information); with -l: show ctime and sort by name;  oth-
              erwise: sort by ctime, newest first

       -C     list entries by columns

       --color[=WHEN]
              colorize the output; WHEN can be 'always' (default if omitted), 'auto', or 'never'; more info below
 Manual page ls(1) line 1 (press h for help or q to quit)
```

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:

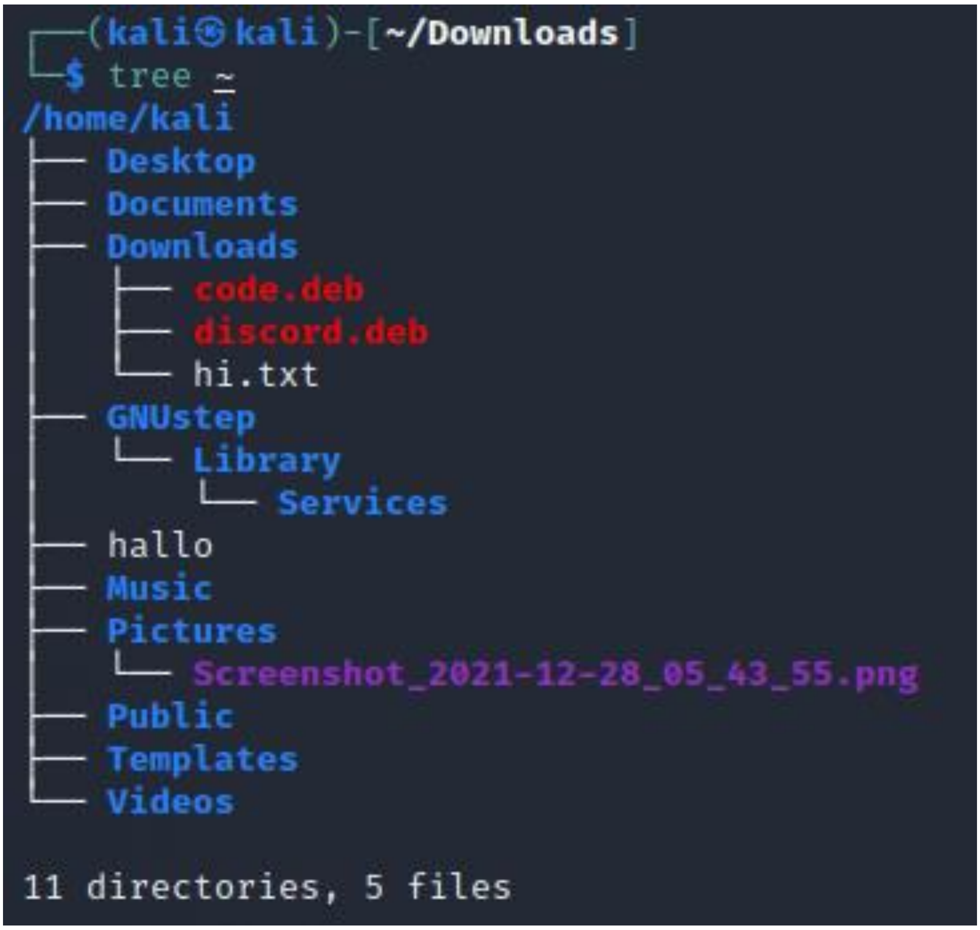
- -a : This will show all files including hidden files | ls -a
- -1 : when adding -1 your folders will be listed vertically | ls -1
- -l : When adding -h the file sizes will be shown we add 1 to it to make the listing more visible | ls -l

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.



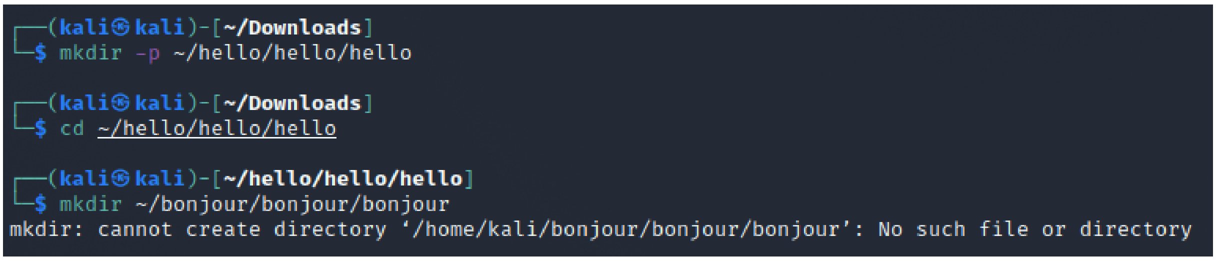
```
┌──(kali㉿kali)-[~/Downloads]
└─$ tree ~
/home/kali
├── Desktop
├── Documents
├── Downloads
│   ├── code.deb
│   ├── discord.deb
│   └── hi.txt
├── GNUstep
│   └── Library
│       └── Services
├── hallo
├── Music
├── Pictures
│   └── Screenshot_2021-12-28_05_43_55.png
├── Public
├── Templates
└── Videos

11 directories, 5 files
```

2.2.4 Create and remove directories

Obviously we can also create and remove directories

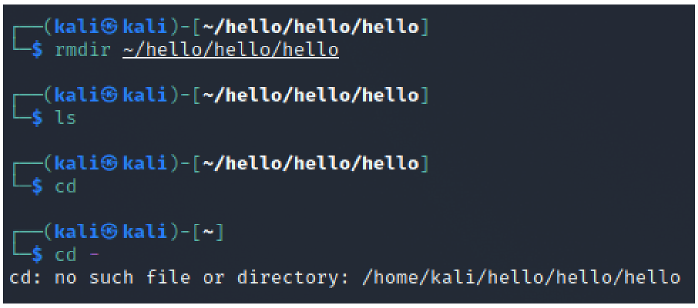
We can simply do mkdir <string> to create a directory with name | mkdir hello.

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.



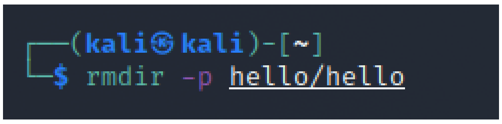
```
┌──(kali㉿kali)-[~/Downloads]
└─$ mkdir -p ~/hello/hello/hello

┌──(kali㉿kali)-[~/Downloads]
└─$ cd ~/hello/hello/hello

┌──(kali㉿kali)-[~/hello/hello/hello]
└─$ mkdir ~/bonjour/bonjour/bonjour
mkdir: cannot create directory ‘/home/kali/bonjour/bonjour/bonjour’: No such file or directory
```

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



```
┌──(kali㉿kali)-[~/hello/hello/hello]
└─$ rmdir ~/hello/hello/hello

┌──(kali㉿kali)-[~/hello/hello/hello]
└─$ ls

┌──(kali㉿kali)-[~/hello/hello/hello]
└─$ cd

┌──(kali㉿kali)-[~]
└─$ cd -
cd: no such file or directory: /home/kali/hello/hello/hello
```

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



```
┌──(kali㉿kali)-[~]
└─$ rmdir -p hello/hello
```

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 -i 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
- i : We can make our rename case insensitive too | rename 's/\.text/.txt/i' *

2.4 Working with file contents

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

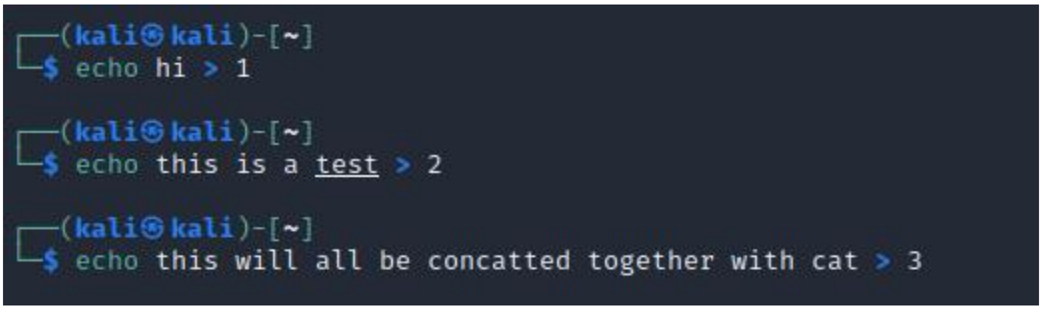
If we wanted to for example only see the first 4 we can do head -4 this will only display the first 4 lines | head -4 /etc/passwd

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.

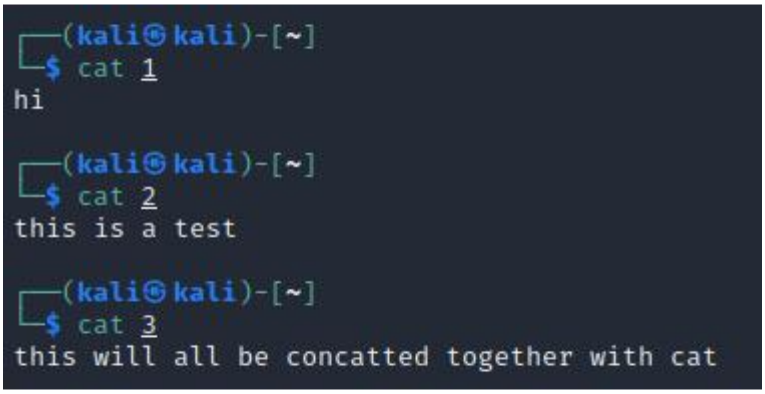
Cat stands for concatenate and then we can obviously concatenate files together, here is an example:

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



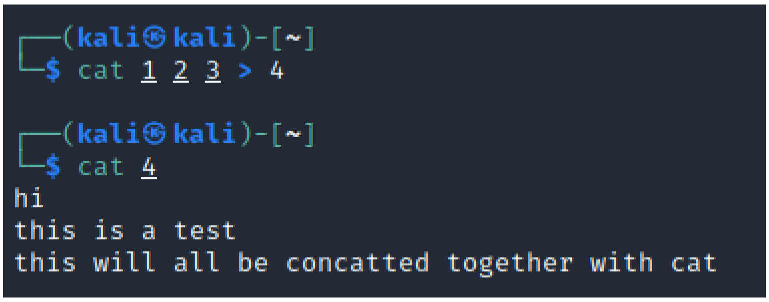
```
┌──(kali㉿kali)-[~]
└─$ echo hi > 1

┌──(kali㉿kali)-[~]
└─$ echo this is a test > 2

┌──(kali㉿kali)-[~]
└─$ echo this will all be concatted together with cat > 3
```

Then we can see the content of those files:



```
┌──(kali㉿kali)-[~]
└─$ cat 1
hi

┌──(kali㉿kali)-[~]
└─$ cat 2
this is a test

┌──(kali㉿kali)-[~]
└─$ cat 3
this will all be concatted together with cat
```

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

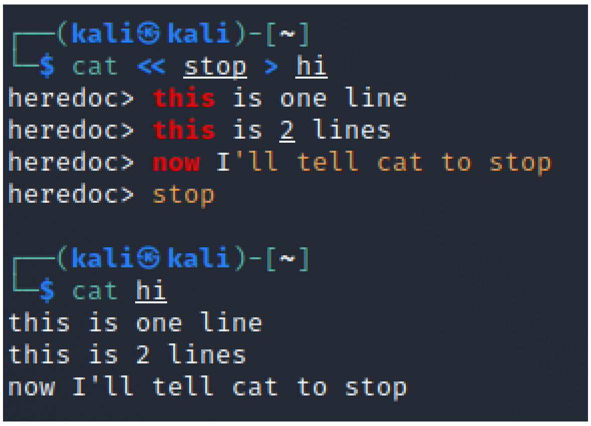


```
┌──(kali㉿kali)-[~]
└─$ cat 1 2 3 > 4

┌──(kali㉿kali)-[~]
└─$ cat 4
hi
this is a test
this will all be concatted together with cat
```

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.



```
┌──(kali㉿kali)-[~]
└─$ cat << stop > hi
heredoc> this is one line
heredoc> this is 2 lines
heredoc> now I'll tell cat to stop
heredoc> stop

┌──(kali㉿kali)-[~]
└─$ cat hi
this is one line
this is 2 lines
now I'll tell cat to stop
```

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

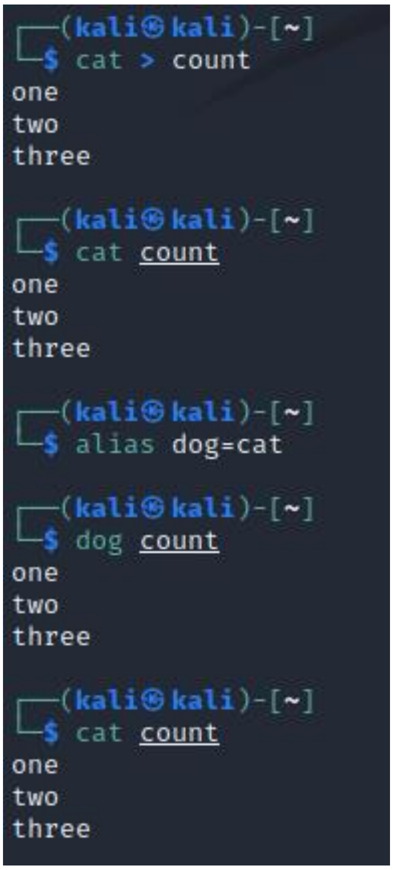
External or building command?

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



```
┌──(kali㉿kali)-[~]
└─$ cat > count
one
two
three

┌──(kali㉿kali)-[~]
└─$ cat count
one
two
three

┌──(kali㉿kali)-[~]
└─$ alias dog=cat

┌──(kali㉿kali)-[~]
└─$ dog count
one
two
three

┌──(kali㉿kali)-[~]
└─$ cat count
one
two
three
```

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

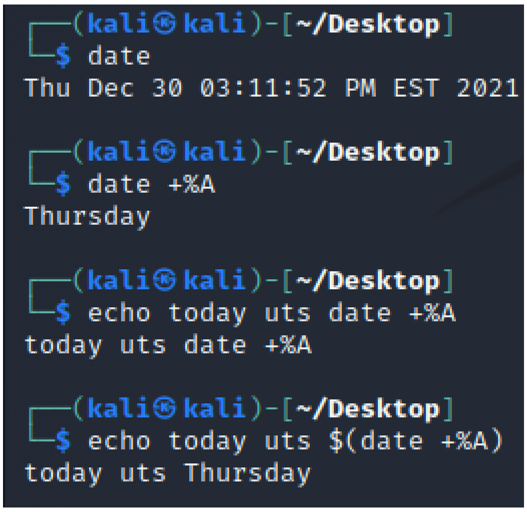
When combining shell variables with other string values we use ${shell variable}<string> instead of shellVar<string> for obvious reasons | echo hello ${prefix}Daan greetings

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



```
┌──(kali㉿kali)-[~/Desktop]
└─$ date
Thu Dec 30 03:11:52 PM EST 2021

┌──(kali㉿kali)-[~/Desktop]
└─$ date +%A
Thursday

┌──(kali㉿kali)-[~/Desktop]
└─$ echo today uts date +%A
today uts date +%A

┌──(kali㉿kali)-[~/Desktop]
└─$ echo today uts $(date +%A)
today uts Thursday
```

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

2.12.1 Introduction

One of the powers of Unix command line is the use of input/output redirection and pipes. In here we will go over redirection of input, output and error streams.

stdin = input stream | stdout = output stream | stderr = error stream

2.12.2 I/O redirection basics

The basics of I/O redirection is quite simple, you take something that gets read to the console and you move it somewhere else with >
| echo This gets written to a file > aFile

When there is nothing being outputted (like with an error) and you write to a file with > the file ends up being empty
| skraaapapa hi > aFile

If we want to append to a file so basically add stuff behind it we can use >>
| echo this gets added behind the content >> aFile

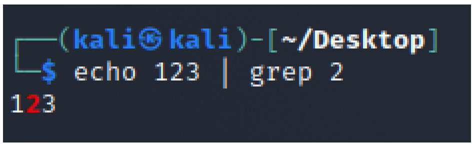
Errors have a separate stream, we can redirect errors with 2>
| bitcoonneeeeeeect 123 2> errorfile

We can use multiple streams in one command
| find / > allfiles 2> /dev/null

If we want to redirect the errors to the same file as the output we can add 2>&1 behind our output, this will use the same stream for error stream as output stream | find / > allfiles 2>&1

2.12.3 Output redirection and pipes

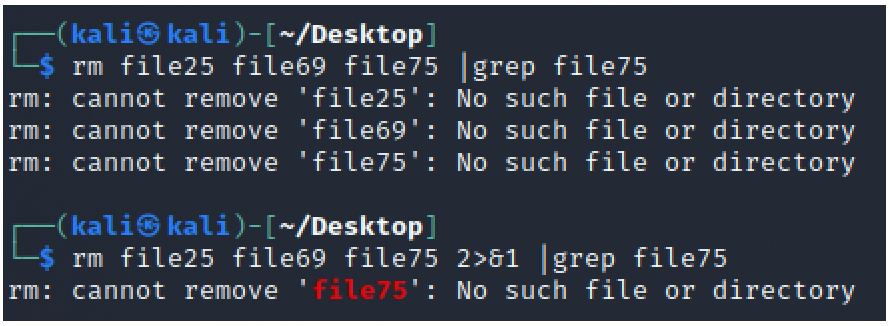
Pipes are basically the things that you do after, for example.
You say
| echo 123 | grep 2
what will happen is it will normally print 123 to the console, this is your output stream but there is a | so instead of printing it will send it to the input stream of the command after |, after the | grep is and to put it simply grep 2 will mark the 2 in the input given by echo and print it out to the console.



```
┌──(kali㉿kali)-[~/Desktop]
└─$ echo 123 | grep 2
123
```

Normally we can not grep error streams but you actually can use input streams when using pipes so what we can do is we can send the error message through the input stream using the 2>&1 we used earlier! For example
| rm file25 file69 file75 | grep file75

This won't work and will give 3 errors while we only want the error from file75 but because grep doesn't take an error stream it will just skip the |. To fix this we can send the error through the input stream
| rm file25 file69 file75 2>&1 | grep file75



```
┌──(kali㉿kali)-[~/Desktop]
└─$ rm file25 file69 file75 |grep file75
rm: cannot remove 'file25': No such file or directory
rm: cannot remove 'file69': No such file or directory
rm: cannot remove 'file75': No such file or directory

┌──(kali㉿kali)-[~/Desktop]
└─$ rm file25 file69 file75 2>&1 |grep file75
rm: cannot remove 'file75': No such file or directory
```

Now lets say we only want the error messages but not the output messages, for this we need to use an extra stream, what we will do is we will move the output stream to an unused stream and move the error stream to the input stream and that stream will work just fine

```
| rm file 42 3>&1 1>&2 2>&3
```

First we place the contents of stream &1 (output stream) in stream 3. Then we put the output from stream 2 (error stream) in stream 1 (output stream) and to top it off place our third pipeline where the output stream is at in stream 2 (error stream). With this we changed places between stream 1 and 2.

It is also possible to join the error stream and output stream this goes with &>

```
| rm file47 &> out
```

We can use the same thing for a pipeline in |&

```
| ls /var/spool/[rc]* |& grep oo
```

2.12.4 Input redirection

We can also do the same for input Ofcorse! This goes with the < sign!

We can simply use < to read for example from a file | echo < test

We can also append with <<

```
| cat << three < count
```

We can also use <<< to directly pass strings to a command, for example if we want to pass hello to Base64

```
| base 64 <<< “hello”
```

The quickest way to clear a file

```
| > file
```

2.13 Filters

Commands made to be used with pipe are often called with filters, these are fairly small and easy to use, I'm pulling a long night to get this done before new year so we are going to go over these fast and efficiently.

cat takes stdin from a string and makes it stdout (takes input prints to console)

```
| cat <<< hello
```

tee is used in between pipelines, when tee gets an stdin it will print stdin to a file and send it through the next pipeline

```
| tac count.txt | tee temp.txt | tac
```

grep is a fun command it will filter out lines containing or not containing a certain input.

```
|cat toolbox.txt | grep hammer
```

Grep has some interesting parameters:

- -v : To search for lines not matching
  ```
  |cat toolbox.txt | grep -v hammer
  ```
- -I : To make it case insensitive
- Ofcorse we can also combine these
  ```
  | cat toolbox.txt | grep -iv HammEr
  ```
- -A1 : Displays 1 line after the result
- -B1 : Displays 1 line before the result
- -C1: Displays 1 line before and after the result
- -e : we can use this to search for multiple matches
  ```
  | cat toolbox.txt | grep -e hammer -e saw
  ```

We can also cut pieces of string away with cut this way we can for example cut away everything except username and userid in /etc/psswd.
| cut -d: -f1,3 /etc/passwd | tail -4

What this does is it will use : as a delimiter instead of TAB
Then it will take field 1 and 3, pipelines it to the tail command and outputs the last 4 lines.

You can also cut on character using c you can specify a range for example for 2-7
| cut -c2-7 /etc/passwd | tail -4

With tr we can translate characters for example e to E
| cat tennis.txt | tr ‘e’ ‘E’
This also has some interesting options:

- Using ranges : | cat tennis.txt tr ‘a-z’ ‘A-Z’
- Changing newline to space | cat tennis.txt | tr ‘\n’ ‘ ’
- -s : Squeeze multiple occurrences of space into 1 space
  | cat tennis.txt | tr -s ‘ ’
- We can also use encrypt text with for example Caesar encoding
  | cat tennis.txt | tr ‘a-z’ ‘defghijklnmopqrstuvwxyzabc’
- -d : we can use this to delete a specific character
  | cat tennis.txt | tr -d e

We can use wc to count words lines characters:

- Counting all | wc tennis.txt
- -l : counting all lines | wc -l tennis.txt
- -w : counting all words | wc -w tennis.txt
- -c : counting all characters | wc -c tennis.txt

We can sort multi line pages with sort

```
| sort tennis.txt
```

With this we can also base our sorting specs on column with -K1

```
| sort -k2 tennis.txt
```

If you want to do a numerical sort you will also need to add -n

```
| sort -n -k3 tennis.txt
```

To remove duplicated we can use uniq, we can also add -c to put the occurrences in front of it

```
| sort music.txt | uniq -c
```

To compare streams we can use comm. It will check if the output is exactly the same, its recommended to sort before checking because otherwise its not identical!

```
| comm -12 list1.txt list2.txt
```

the digits 1 2 point out which columns should not be displayed

sed (stream editor) can perform editing functions in the stream using regular expressions.

```
| echo level5 | sed ‘s/5/42/’
```

If we want to globally change this we can add ‘g’ behind the regular expression | echo level5 | sed ‘s/5/42/g’

If we want to delete an occurrence we can add ‘d’

```
| echo level5 | sed ‘/5/d’
```

Here are some examples to try:

- Who | wc – l
- Who | cut -d’ ‘ -f1 | sort
- who | cut -d' ' -f1 | sort | uniq
- grep bash /etc/passwd | cut -d: -f1

2.14 Basic Unix tools

In here we will go over basic commands to find or locate files and to compress files. The tools here are not considered filters

find is an useful command to start off a pipeline, this finds all files in the given directory | find /etc
This has some interesting options:

- -name : find all .conf files in the current directory (. Means all)
  | find . -name "*.conf"
- -type f : find all .conf files | find . -type f -name "*.conf"
- -type d: Find all files of type directory that end in .bak
  | find /data -type d -name "*.bak"
- -newer : find all files newer than given path
  | find . -newer file24
- We can also use find to execute another command in the file (the command we execute is cp <file> /backup/\) found
  | find /data -name "*.odf" -exec cp {} /backup/ \;
- -ok : We can also only execute it if we get confirmation
  | find /data -name "*.odf" -ok rm {} \;

locate is a different tool as find it uses indexes to locate files, this is faster but in order to do so you'll need to update the database before using it best
| sudo updated
| locate file

We used date before but we will now go into a little more detail:

- A date string can be customised to display the format of choice, when doing man date you'll find all the options
  | date +'%A %d-%m-%Y'
- +%s : we can use this to display time in seconds since 1969 as this is when time started being calculated in seconds for unix
  | date +%s
- We can also increment the time with a simple + so if we want to check when the time will reach two thousand million
  | date -d '1970-01-01 + 2000000000 seconds'

We can also generate a calendar with cal, this displays a calendar of the current month, but we can also specify a month we want to see
| cal 2 1970

We can use the sleep command in scripts to wait a number of seconds | sleep 5

We can use the time command to display how long it takes for a command to execute, we can for example check how long the command date of sleep takes named real
| time date
| time sleep 5
| time find / > myFiles 2> dev/null

We can zip/compress a file or directory with gzip
| gzip text.txt

We can unzip/decompress a file or directory with gunzip
| gunzip text.gz

We can use zcat and zmore to view content of a zipped file

```
| zcat text.gz
| zmore text.gz
```

We can also compress with bzip2 and bunzip2 this works the same as gzip but it takes longer and compresses better

```
| bzip2 text.txt
| bunzip text.gz
```

We have to use bzcat or bzmore to read these files

```
| bzcat text.gz
| bzmore text.gz
```

We can use tar to archive/extract a directory structure
We can do this for our home directory

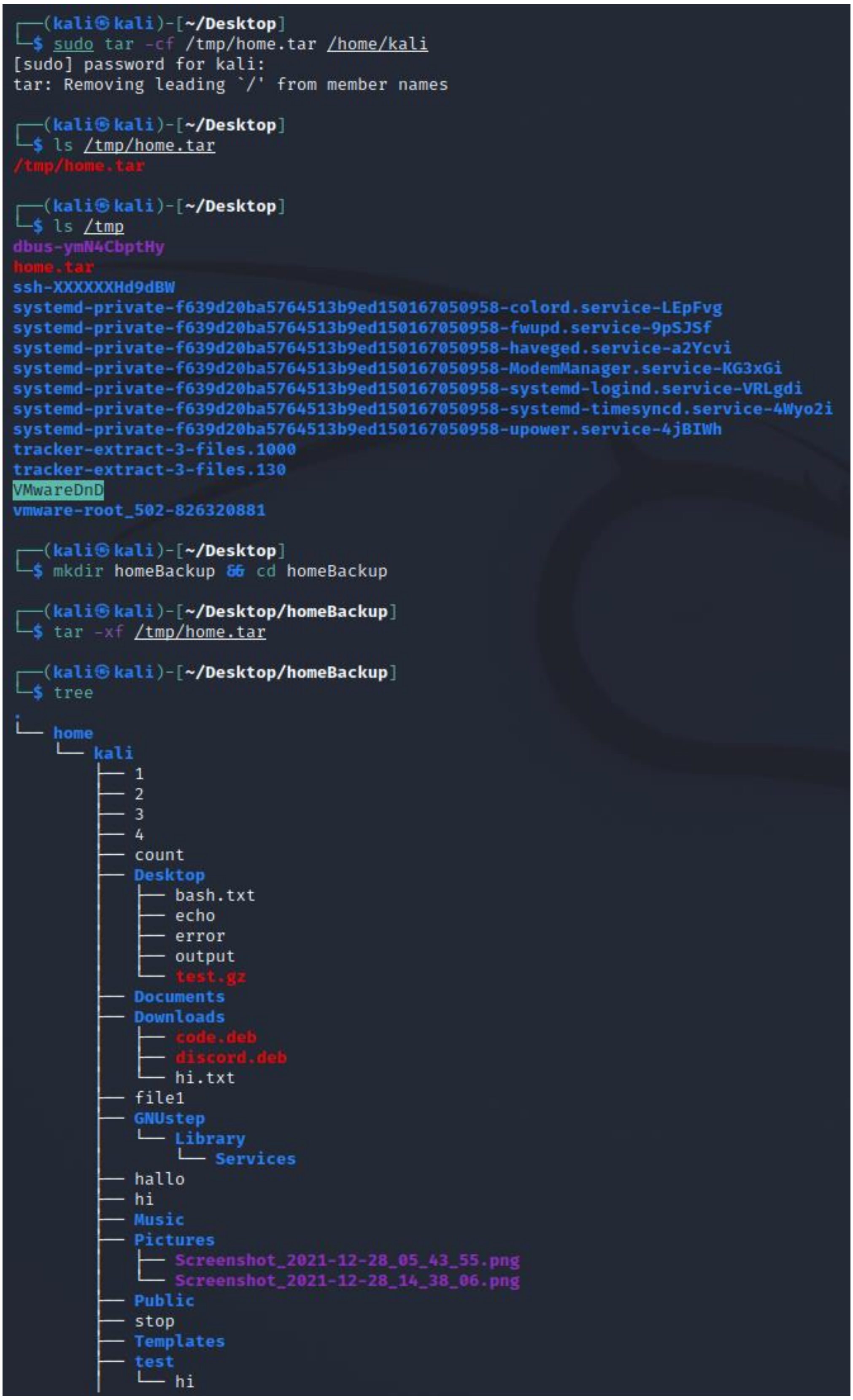
```
| sudo tar -cf /tmp/home.tar /home/kali
```

What we did here is, we took our home folder and compressed it into a .tar file and send it to /tmp/home.tar

We can now extract this folder and see what’s inside

```
| tar -xf /tmp/home.tar
```

Here is the process in code



```
┌──(kali㉿kali)-[~/Desktop]
└─$ sudo tar -cf /tmp/home.tar /home/kali
[sudo] password for kali:
tar: Removing leading `/' from member names

┌──(kali㉿kali)-[~/Desktop]
└─$ ls /tmp/home.tar
/tmp/home.tar

┌──(kali㉿kali)-[~/Desktop]
└─$ ls /tmp
dbus-ymN4CbptHy
home.tar
ssh-XXXXXXHd9dBW
systemd-private-f639d20ba5764513b9ed150167050958-colord.service-LEpFvg
systemd-private-f639d20ba5764513b9ed150167050958-fwupd.service-9pSJSf
systemd-private-f639d20ba5764513b9ed150167050958-haveged.service-a2Ycvi
systemd-private-f639d20ba5764513b9ed150167050958-ModemManager.service-KG3xGi
systemd-private-f639d20ba5764513b9ed150167050958-systemd-logind.service-VRLgdi
systemd-private-f639d20ba5764513b9ed150167050958-systemd-timesyncd.service-4Wyo2i
systemd-private-f639d20ba5764513b9ed150167050958-upower.service-4jBIWh
tracker-extract-3-files.1000
tracker-extract-3-files.130
VMwareDnD
vmware-root_502-826320881

┌──(kali㉿kali)-[~/Desktop]
└─$ mkdir homeBackup && cd homeBackup

┌──(kali㉿kali)-[~/Desktop/homeBackup]
└─$ tar -xf /tmp/home.tar

┌──(kali㉿kali)-[~/Desktop/homeBackup]
└─$ tree
.
└── home
    └── kali
        ├── 1
        ├── 2
        ├── 3
        ├── 4
        ├── count
        ├── Desktop
        │   ├── bash.txt
        │   ├── echo
        │   ├── error
        │   ├── output
        │   └── test.gz
        ├── Documents
        ├── Downloads
        │   ├── code.deb
        │   ├── discord.deb
        │   └── hi.txt
        ├── file1
        ├── GNUstep
        │   └── Library
        │       └── Services
        ├── hallo
        ├── hi
        ├── Music
        ├── Pictures
        │   ├── Screenshot_2021-12-28_05_43_55.png
        │   └── Screenshot_2021-12-28_14_38_06.png
        ├── Public
        ├── stop
        ├── Templates
        ├── test
        │   └── hi
```

2.15 Regular expressions

2.15.1 introduction

In here we will go over regular expressions, honestly a pain in the ass to learn but a really powerful thing to master in Linux!

There are 3 different versions of regular expression syntax:

- BRE: Basic Regular Expressions
- ERE: Extended Regular Expressions
- PRCE: Perl Regular Expressions

Depending on the tool being used more of these syntaxes can be used.

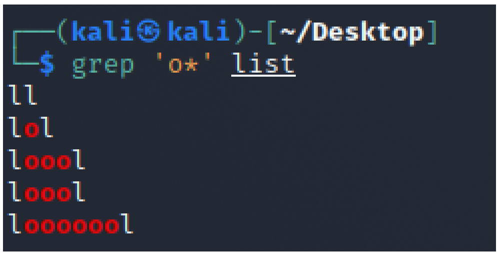
2.15.2 expressions

We already went over grep but there is much more depth to it with regular expressions.
We can grep for a single or multiple characters and only the lines containing the characters get returned:
| grep ai names.txt

In grep we can also use or which will check if one or another is in the line with -E | grep -E ‘i|a’ list
We can also use this as -e | grep -e 'i' -e 'a' list

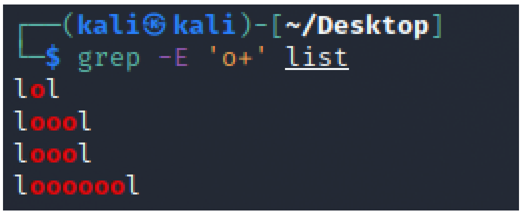
With BRE (Basic regular expressions) the meta characters ?, +, {, |, (and) lose their special meaning. You need to use the backslashed version to interpret right, like \?, \+, \{, \|, \(and \)
| grep 'i\|a' list

An asterisk * signifies zero, one or more occurrences of the previous character



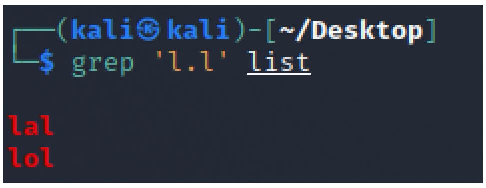
```
┌──(kali㉿kali)-[~/Desktop]
└─$ grep 'o*' list
ll
lol
loool
loool
looooool
```

A plus sign signifies one or more occurrences of the previous char



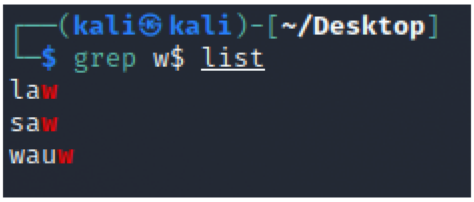
```
┌──(kali㉿kali)-[~/Desktop]
└─$ grep -E 'o+' list
lol
loool
loool
looooool
```

A dot . signifies any character



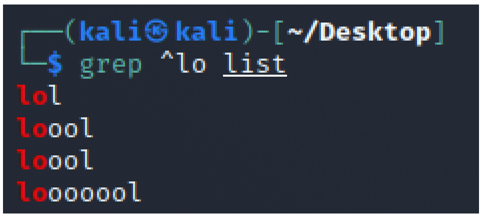
```
┌──(kali㉿kali)-[~/Desktop]
└─$ grep 'l.l' list

lal
lol
```

We can also check for a line ending with a dollar sign $ (Watch out, a space is also a character)



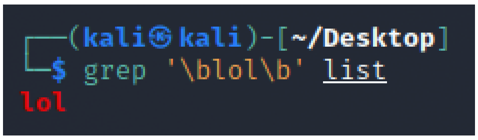
```
┌──(kali㉿kali)-[~/Desktop]
└─$ grep w$ list
law
saw
wauw
```

We can match the start of a string with ^



```
┌──(kali㉿kali)-[~/Desktop]
└─$ grep ^lo list
lol
loool
loool
looooool
```

If we want to search well we will use \b with this we can specify a word and only that word will be found in a sentence or listing



```
┌──(kali㉿kali)-[~/Desktop]
└─$ grep '\blol\b' list
lol
```

We can also use -w in the same way to only search words

Its important to note that you best always quote dollar signs for a regex like a$ so it won’t get confused with the shell

On Debian Linux rename is a perl script and so uses perl regular expressions, a complete manual can be found after installing perldoc with | sudo apt install perldoc
with the command | perldoc perlrequick

We already went over using rename but lets quickly refresh the basic syntax of changing the extension of all files in a directory
| rename ‘s/.txt/.png’ *

Or we can say that we want all files that end on .text to be changed to .txt | rename ‘s/text/txt/’ .text

Replace only changes the first occurrence in a string so if you have multiple matches only the first match will do

We can fix this by adding the g for global, this will change all occurrences | rename -n ‘s/TXT/txt/g’ a TXT.TXT

We can also use case insensitive replaces with i
| rename ‘s/\.text/.txt/i’ *

We can also use a $ sign to specifically rename the extension
| rename ‘s/\.txt$/.TXT’ *

We have sed to edit streams, to put it really simple we can echo Sunday in a pipeline and change it to Monday

```
| echo Sunday | sed ‘s/Sun/Mon/’
```

We can also replace the / by a : or a _ or a | but that has rarely any use

While sed is mostly used in streams we can also use it in-place as it just reads input streams and so this also works for reading a file

```
| sed -i ‘s/Sun/Mon/’ today
```

We can use ampersand & to reference the searched and found string we can use this to for example search and double to occurrence of a string | echo Sunday | sed ‘s/Sun/&&/’
Or add random s’es in front

```
| echo Sunday | sed ‘s/Sun/SSSSSSSSSSS&/’
```

This way we can also add a # in front of commands

We can also do back referencing this way we can reference a group section and reuse it for example

```
| echo Sunday | sed ‘s/\(Sun\)/\1ny/’
```

To reference we are using the \ but ofcorse we still need to use the / for our regex to work.
First we say we will be back reference Sun by doing \(Sun\)
Then we will reference to Sun again with \1 and put ny behind it, if we where to hardcode this we would do | sed ‘s/Sun/Sunny’

The same also works for multiple

```
| echo 2024-04-01 | sed 's/\(....\)-\(..\)-\(..\)/\1+\2+\3/'
```

Here we will echo a date and split all parts up and then replace the original date with the dates with a + in between instead of a –

Using \s we can reference whitespace or a tab

```
| echo -e ‘today\tis\twarm’ | sed ‘s/\s/ /g
```

What we do here is, we look for white spaces and globally change them to 1 space

A question mark signs that the previous character is optional for example | cat list | sed -r ,’s/ooo?/A/’
What it does is it will search for 3 o’s but the 3th o is optional so 2 o’s is enough and changes those with A

We can also specify 3 o’s in another way then ‘ooo’ using o{3}

```
| cat list | sed -r ‘s/o{3}/A/’
```

We can also say it can be between min and max

```
| cat list | sed -r ‘s/o{2,3}/A/’
```

A cool thing is that these things also work with the bash history as these are also just strings!

2.16 Working with vi

Vi is an editor installed on almost every Unix system, Linux very often installs vim which is similar but improved. Its really powerful but difficult to get started with

We can simply start the vi editor by writing vi in the command line

```
| vim
```

```
| vim <filename>
```

We can use Esc to enter command mode and insert mode

When opening vi you start in command mode, when in command mode there are a couple of commands to take a note of to get started:

- a : start typing after the current character
- A : start typing at the end of the current line
- i : start typing before the current character
- I : start typing at the start of the current line
- o : start typing on a new line after the current line
- O : start typing on a new line before the current line

If we have typed something in the vi editor we can also replace and delete characters:

- x : delete the character below the cursor
- X : delete the character before the cursor
- r : replace the character below the cursor
- q : paste after the cursor (here the last deleted character)
- xp : switch two characters

We can ofcorse also undo and repeat:

- u : Undo the last action
- . : Repeat the last action

We can also cut, copy and paste:

- dd : Cut current line
- yy : Copy current line (Yank Yank)
- p : Paste after current line
- P : Paste before current line

Start and end of line:

- 0 or ^ : Jump to start of current line
- $: Jump to end of current line
- d0 : Delete until start of line
- d$: delete until end of line

Join two lines:

- J : Join two lines
- yyp : Duplicate a line
- ddp : Switch two lines

Words:

- w : Forward one word
- b : Back one word
- 3w : Forward 3 words
- dw : Delete one word
- yw : Yank (copy) one word
- 5yb : Yank five words back
- 7dw : Delete seven words

Save and exit, for this we will also work with a semicolon at the start :, what you do is, type : then write the command and then press enter:

- :w : Save (write)
- :w name : Save as name
- :q : Quit
- :wq : Save and quit
- ZZ : Save and quit (This is without :)
- :q! : Quit without saving
- :w! : Save when writing to non-writeable file

I wouldn’t be able to pass my exams without Ctrl+F, luckily vi has some similar tools, for this we will use / and ? the same way as we used : in save and exit:

- /<string> : forward search for <string>
- ?<string> : backward search for <string>
- n : go to next occurrence in search
- /^<string> : Forward search <string> at beginning of line
- /<string>$: Forward search <string> at end of line
- /l[oaeu]l : search for lol, lal, lel, lul
- /\<he\> : only search for he and not hem or lhe

This also use regular expressions

There is also replace functionality:

- :4,8 s/foo/bar/g : replace foo with bar on lines 4 to 8
- :1,$ s/foo/bar/g : replace foo with bar on all lines

We can also use vi to read files:

- :r <string> : (read) file <string> and paste contents
- :r !cmd : execute cmd and paste its output

We can also use abbreviations:

- :ab str long string : Abbreviate str to be ‘long string’
- :una str : we can unabbreviate str

We can also map or keys, for example if we want F6 to toggle between set number and set nonumber we can do:

| :map <F6> :set number!<bar>set number?<CR>

There also some settings options that might be interesting, we can set these by doing | vim ~/.vimrc

- :set number (also try :se nu)
- :set nonumber
- :syntax on
- :syntax off
- :set all (list all options)
- :set tabstop=8
- :set tx (CR/LF style endings)
- :set notx

2.17 Users

Now we will go everything User related

Do get our name we can use whoami

```
| whoami
```

For info on who logged into your system who

```
| who
```

w shows who is logged in and what they are doing

```
| w
```

id gives your user id, primary group id and list of groups you are in

```
| id
```

If you want to run a shell on another user you can use su

```
| su daan
```

If you aren’t logged in as the root you’ll need a password to do so

if we want to log into the root we can do su – or just su

```
| su –
```

If we want to run a program as an other user we need to put sudo in front

| sudo /usr/sbin/useradd -m root

In some Linux distributions like Linux root doesn’t have a password set so you can’t log in to root yet, we can give ourself these at
| sudo vim /etc/sudoers

When we did that we can use | sudo su –
We can use CTRL+D to leave the root

The local user database on Linux is: /etc/passwd

The root user aka. superuser is the most powerful account on Linux systems, it can almost do anything.

We can easily add users using this command
| useradd -m -d /home/daan -c "Daan Detre" Daan

The example below shows how to add a user named daan (last parameter) and at the same time forcing the creation of the home directory (-m), setting the name of the home directory (-d), and setting a description (-c).

Many Linux distributions have a file called /etc/default/useradd with default parameters when creating an user, we can see those with cat or | useradd -D

we can also delete users from our system
| userdel -r daan

We can also use usermod to modify our users, in this example we change the description

```
| usermod -c 'suuuppeeeerrr daan' daan
```

If we forgot to add our home directory and it wasn't created by default we can do it manually with mkdir en use chmod and chown to set permissions (More about that in the next 2 chapters)

```
| mkdir /home/daan
| chown daan:daan /home/daan
| chmod 700 /home/daan
```

We can also delete home directories same way as users

```
| userdel -r daan
```

There is also a login shell which is specified in /etc/passwd

An user can change its login shell with chsh <Index> we can get the list of shells with | chsh -l | cat /etc/shells

and change shells with | chsh -s /bin/ksh

We can set/change passwords with the passwd command

```
| passwd
```

All these passwords are stored in the shadow file we can see these via root at /etc/shadow

```
| cat /etc/shadow
```

The recommended way of adding an user is to create the user and add the password with passwd

The /etc/login.defs file contains some default settings for user passwords like password aging and length settings.

```
| grep PASS /etc/login.defs
```

We can also set an expiration date on passwords, with -l we can list all settings

```
| chage -l daan
```

If we want to disable a password we can use usermod

```
| usermod -L daan
```

We can then check using grep and you'll see that just a ! has been added in front of the password

```
| grep daan /etc/shadow | cut -c1-70
```

The root can then reenable the password with

```
| usermod -U daan
```

You can still manually edit the /etc/passwd or /etc/shadow with vipw

```
| vipw/passwd
```

Each system has a system profile has a path we can grep all of these

```
| grep PATH /etc/profile
```

Here are some more interesting files to know:

- ~/.bash_profile : When this file exists in the home directory bash will source it
- ~/.bash_login : If ~/bash_profile doesn't exist then bash will look for ~/bash_login and source it
- ~/.profile : if bash_profile and bash_login don't exist it will check the existence of ~/profile
- ~/.bashrc : often sourced by other scriptswe can check what it does with | cat /home/kali/.bashrc
- ~/.bash_logout : this ghets used when exiting bash it also claers the console screen

2.18 Groups

Groups are used to set permissions on a group level instead of giving each individual the same permissions

We can easily create groups with groupadd

```
| groupadd soccer
```

Membership of groups are defined by the /etc/group file

```
| cat /etc/group
```

An user can see what group it belongs to with groups

```
| groups
```

We can use usermod and useradd to modify secondary group members

```
| usermod -a -G soccer daan
```

We can only use usermod to specify a primary login group for a user, the primary group will become groupower of every file and folder

```
| usermod -g soccer daan
```

We can use gpasswd to modify groups

To add an user to a group we can do

```
| gpasswd -a daan soccer
```

To remove a user from a group we can use

```
| gpasswd -d daan soccer
```

You can change the group name with groupmod

```
| groupmod -n soccer football
```

We can delete a group with groupdel

```
| groupdel football
```

We can delegate control of a group membership to another user with gpasswd

```
| gpasswd -A daan soccer
```

Group information can be found in /etc/gshadow

```
| cat /etc/gshadow
```

We can empty the group administrators with setting an empty string where we would normally set control

```
| gpasswd -A “” soccer
```

We can set a child shell with a new temporary primary group using the newgrp command

```
| newgrp soccer
```

We can use vigr to manually edit the /etc/group file but its recommended not to do it this way for inexperienced administrators

2.19 Standard file permissions

This is some kind of follow up to the groups part in which we are going to go about file ownership and file permissions, there are some commands we went over in earlier parts so I'll just mention these quickly

If we want to see more details like each files user owner we can already see it with

```
| ls -lh
```

see all local user accounts

```
| cut -d: -f1 /etc/passwd | column
```

we can change the group owner with chgrp

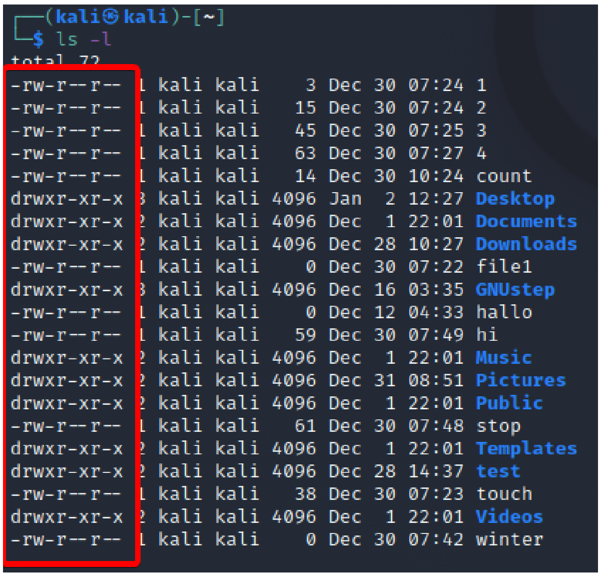
```
| chgrp snooker file2
```

The user owner of a file can be changed with chown command.

```
| chown daan soccer
```

When you use ls -l, for each file you can see ten characters before the user and group owner. The first character tells us the type of file. Regular files get a -, directories get a d, symbolic links are shown with an l, pipes get a p, character devices a c, block devices a b, and sockets an s.

first character	file type
-	normal file
d	directory
l	symbolic link
p	named pipe
b	block device
c	character device
s	socket



```
┌──(kali㉿kali)-[~]
└─$ ls -l
total 72
-rw-r--r-- 1 kali kali    3 Dec 30 07:24 1
-rw-r--r-- 1 kali kali   15 Dec 30 07:24 2
-rw-r--r-- 1 kali kali   45 Dec 30 07:25 3
-rw-r--r-- 1 kali kali   63 Dec 30 07:27 4
-rw-r--r-- 1 kali kali   14 Dec 30 10:24 count
drwxr-xr-x 3 kali kali 4096 Jan  2 12:27 Desktop
drwxr-xr-x 2 kali kali 4096 Dec  1 22:01 Documents
drwxr-xr-x 2 kali kali 4096 Dec 28 10:27 Downloads
-rw-r--r-- 1 kali kali    0 Dec 30 07:22 file1
drwxr-xr-x 3 kali kali 4096 Dec 16 03:35 GNUstep
-rw-r--r-- 1 kali kali    0 Dec 12 04:33 hallo
-rw-r--r-- 1 kali kali   59 Dec 30 07:49 hi
drwxr-xr-x 2 kali kali 4096 Dec  1 22:01 Music
drwxr-xr-x 2 kali kali 4096 Dec 31 08:51 Pictures
drwxr-xr-x 2 kali kali 4096 Dec  1 22:01 Public
-rw-r--r-- 1 kali kali   61 Dec 30 07:48 stop
drwxr-xr-x 2 kali kali 4096 Dec  1 22:01 Templates
drwxr-xr-x 2 kali kali 4096 Dec 28 14:37 test
-rw-r--r-- 1 kali kali   38 Dec 30 07:23 touch
drwxr-xr-x 2 kali kali 4096 Dec  1 22:01 Videos
-rw-r--r-- 1 kali kali    0 Dec 30 07:42 winter
```

The 9 character behind the first are all permissions, these are the potential ones

permission	**on a file**	**on a directory**
r (read)	read file contents (cat)	read directory contents (ls)
w (write)	change file contents (vi)	create files in (touch)
x (execute)	execute the file	enter the directory (cd)

The position in which the permissions are sed are important too

| -rwxr-xr--

position	**characters**	**function**
1	-	this is a regular file
2-4	rwx	permissions for the **user owner**
5-7	r-x	permissions for the **group owner**
8-10	r--	permissions for **others**

We can set permission using chmod:

- read perm | chmod g-r permissions.txt
- execute perm| chmod u+x permissions.txt
- remove read perm | chmod o-r permissions.txt
- Gives all of them write perm | chmod a+w permissions.txt
  - a isn't actually needed | chmod +x permissions.txt
- Explicit permissions | chmod u=rw permissions.txt
- Combinations | chmod u=rw,g=rw,o=r permissions.txt
- Combinations | chmod u=rwx,ug+rw,o=r permissions.txt

There is also an octal way to set permissions:

binary	**octal**	**permission**
000	0	---
001	1	--x
010	2	-w-
011	3	-wx
100	4	r--
101	5	r-x
110	6	rw-
111	7	rwx

This makes 777 equal to rwxrwxrwx and by the same logic, 654 mean rw-r-xr-- . The chmod command will accept these numbers.

When creating a file or directory, a set of default permissions are applied. These default permissions are determined by the umask. The umask specifies permissions that you do not want set on by default. You can display the umask with the umask command.

```
| umask -S
| umask
```

A file is also not executable by default. Newly created files are never executable by default. You have to explicitly do a chmod +x to make a file executable.

We can use -m with mkdir to set permissions upfront

```
| mkdir -m 777 Public
```

To preserve permissions and time stamps from source files, use cp -p.

```
| sudo cp -p file* cpp
```