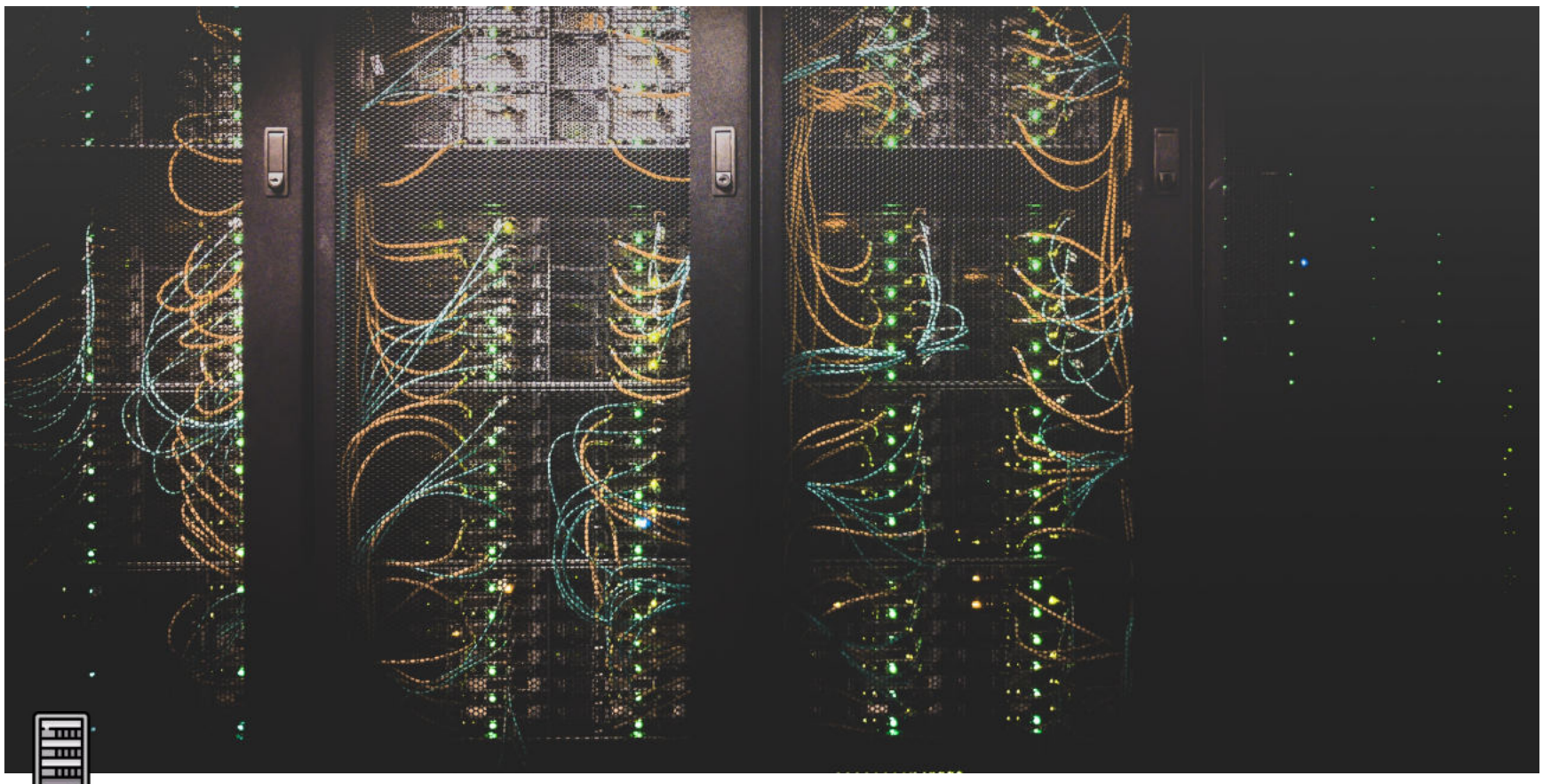


Launching a Virtual Machine

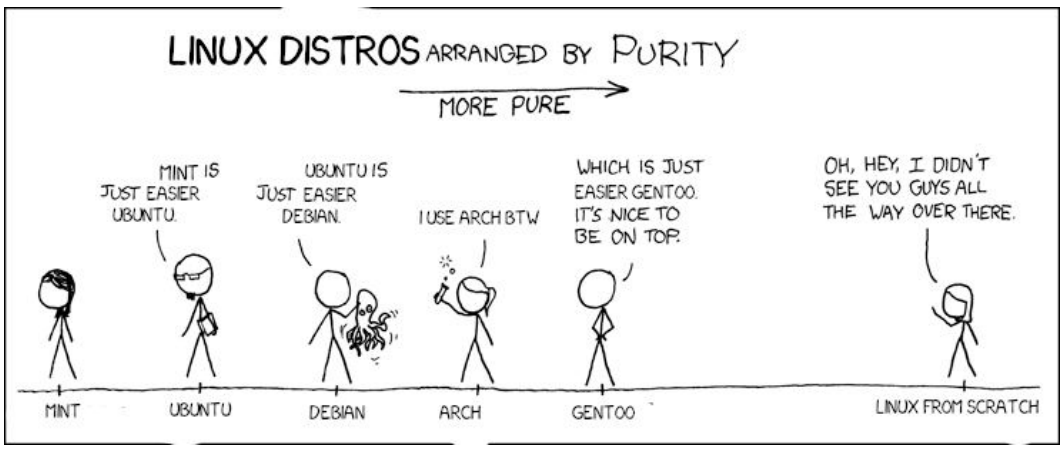
Type	📒 Lecture
Date	@December 22, 2021
Lecture #	1
Lecture URL	https://youtu.be/PHrN7yp1AJw
Notion URL	https://21f1003586.notion.site/Launching-a-Virtual-Machine-e493cecce6a743a9b37516d196c07c1d
Week #	1

Why do we need a Virtual Machine (VM)?

- Laptops usually come pre-installed with Windows
  - Unless, of course, you are an 🍎 person
- We wish to try Linux almost natively, without removing the existing OS
  - Also considering the fact that we do not wish to dual boot

Requirements

- An .iso image of the operating system we want
  - Ubuntu 20.04 is recommended, or just use arch btw



    - Can be downloaded here
- A Hypervisor
  - A **hypervisor**, also known as a virtual machine monitor or VMM, is software that creates and runs virtual machines (VMs). A hypervisor

allows one host computer to support multiple guest VMs by virtually sharing its resources, such as memory and processing.

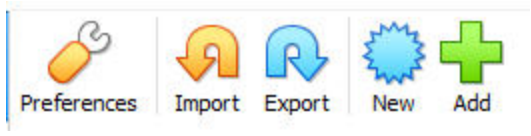
Source: https://www.vmware.com/topics/glossary/content/hypervisor.html

- Oracle VirtualBox
- VMWare Workstation Player
- or just use Windows Subsystem for Linux
- Atleast 20GB free space for the VM
- Some RAM ¯_(ツ)_/¯
  - 8GB+ recommended

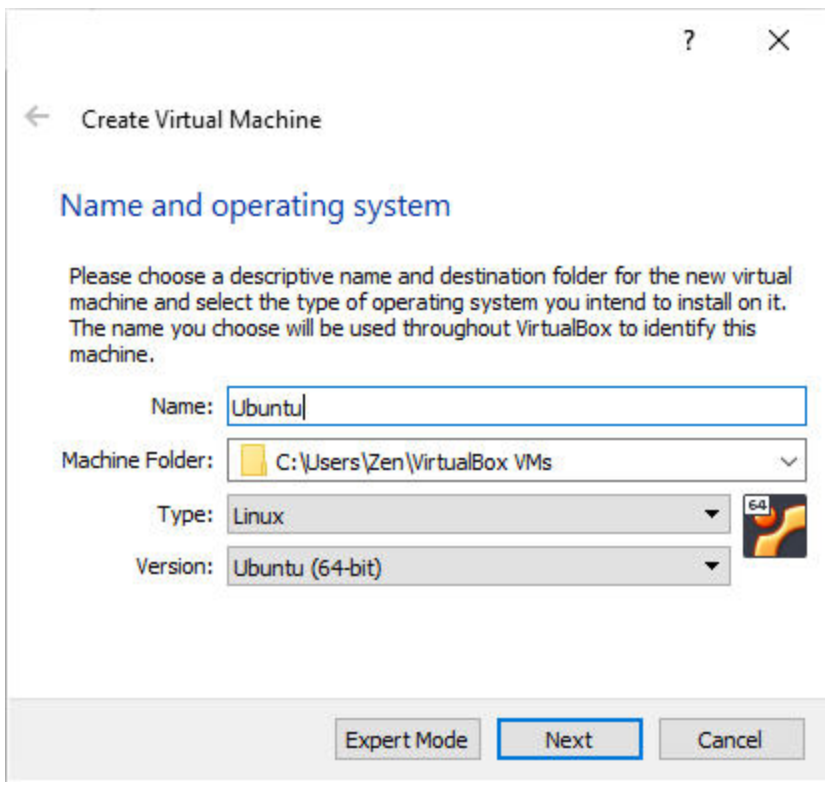
Steps

- Download the Ubuntu .iso file from https://ubuntu.com/download/desktop
- Download either VirtualBox or the VMWare Workstation Player
  - Install them
- Open VirtualBox / VMWare Workstation Player

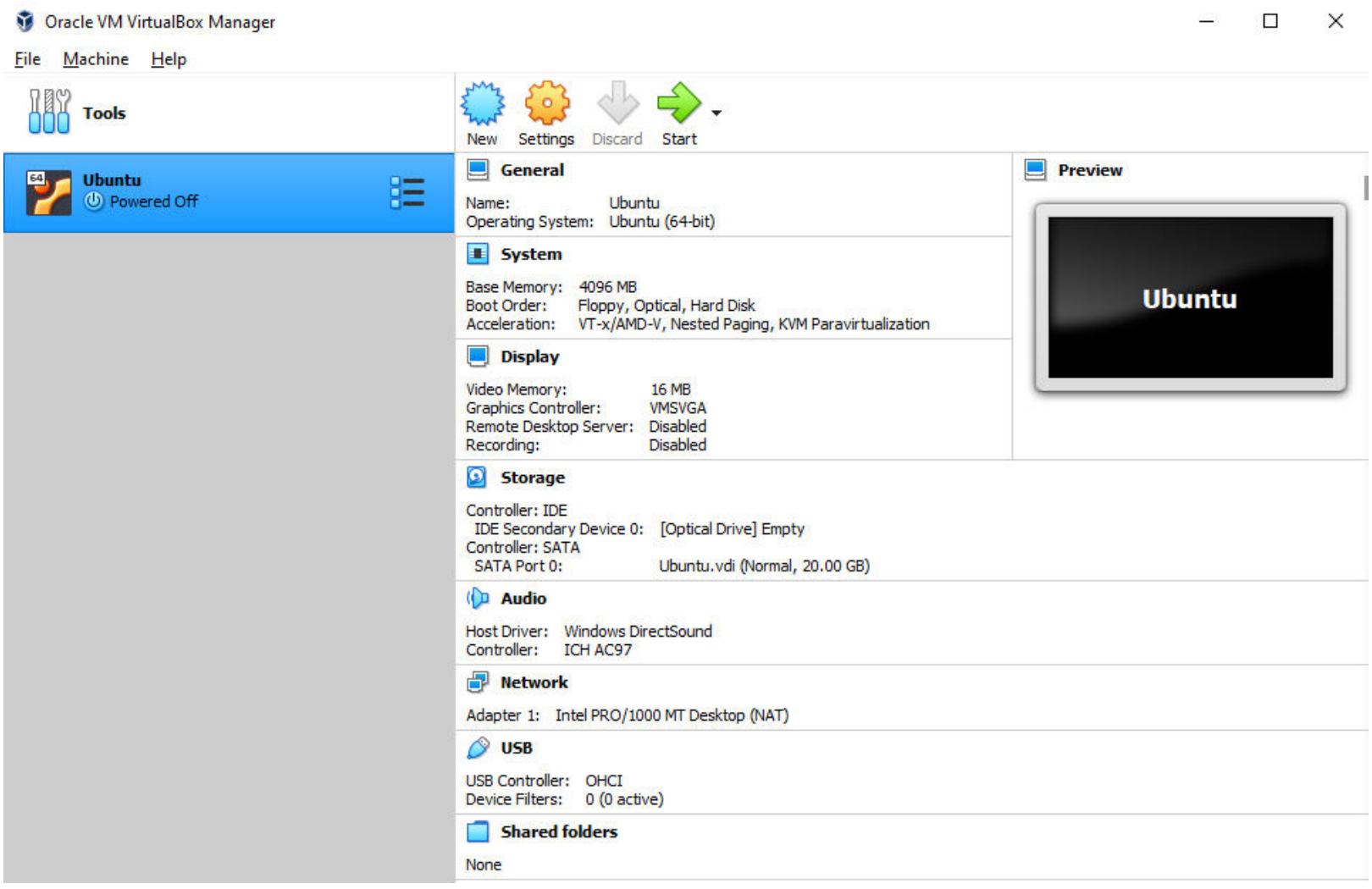
  (I will be using VirtualBox)
  - Click on the “New Button to create a new VM”



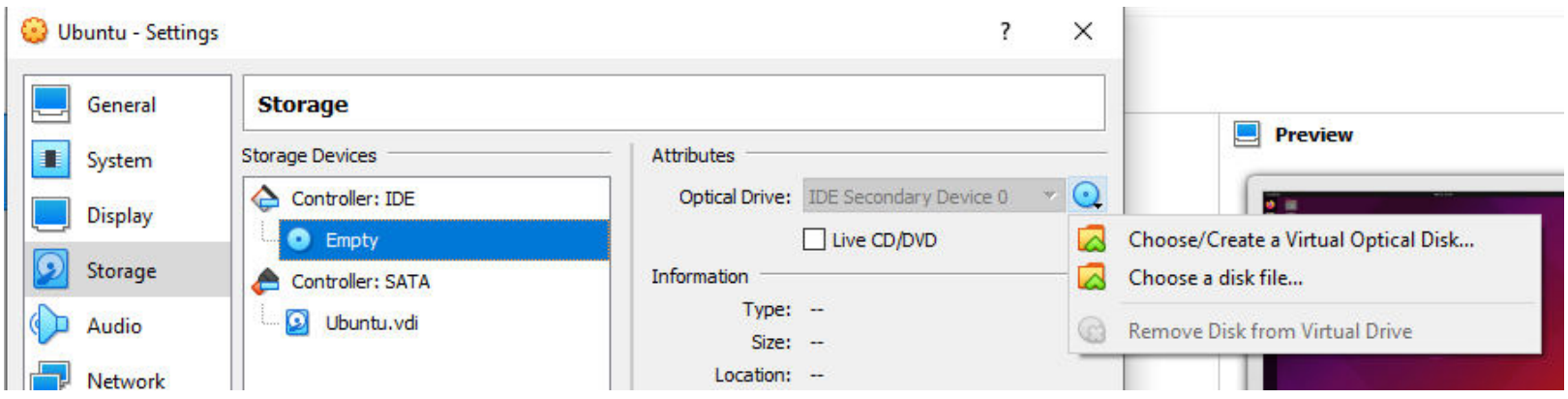
  - Add a name and choose the OS type and version



  - Adjust the RAM and Storage as per your liking, make sure to keep the minimum specs
  - Select the VM on the left menu and go to settings

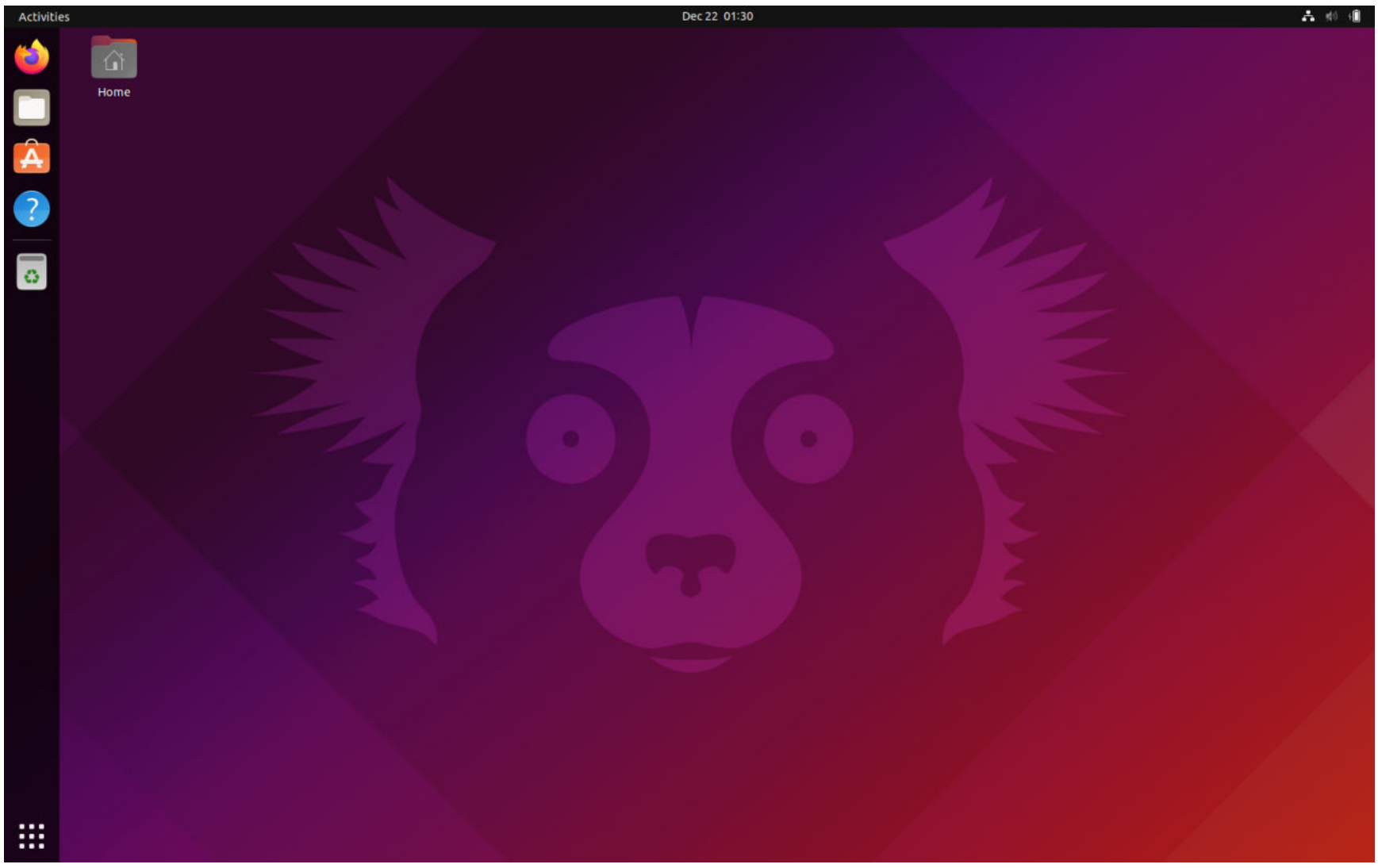


- Go to Storage → Storage Devices → Controller → Empty → Click on the CD icon and choose "**Choose/Create a Virtual Optical Disk...**" and choose your .iso file

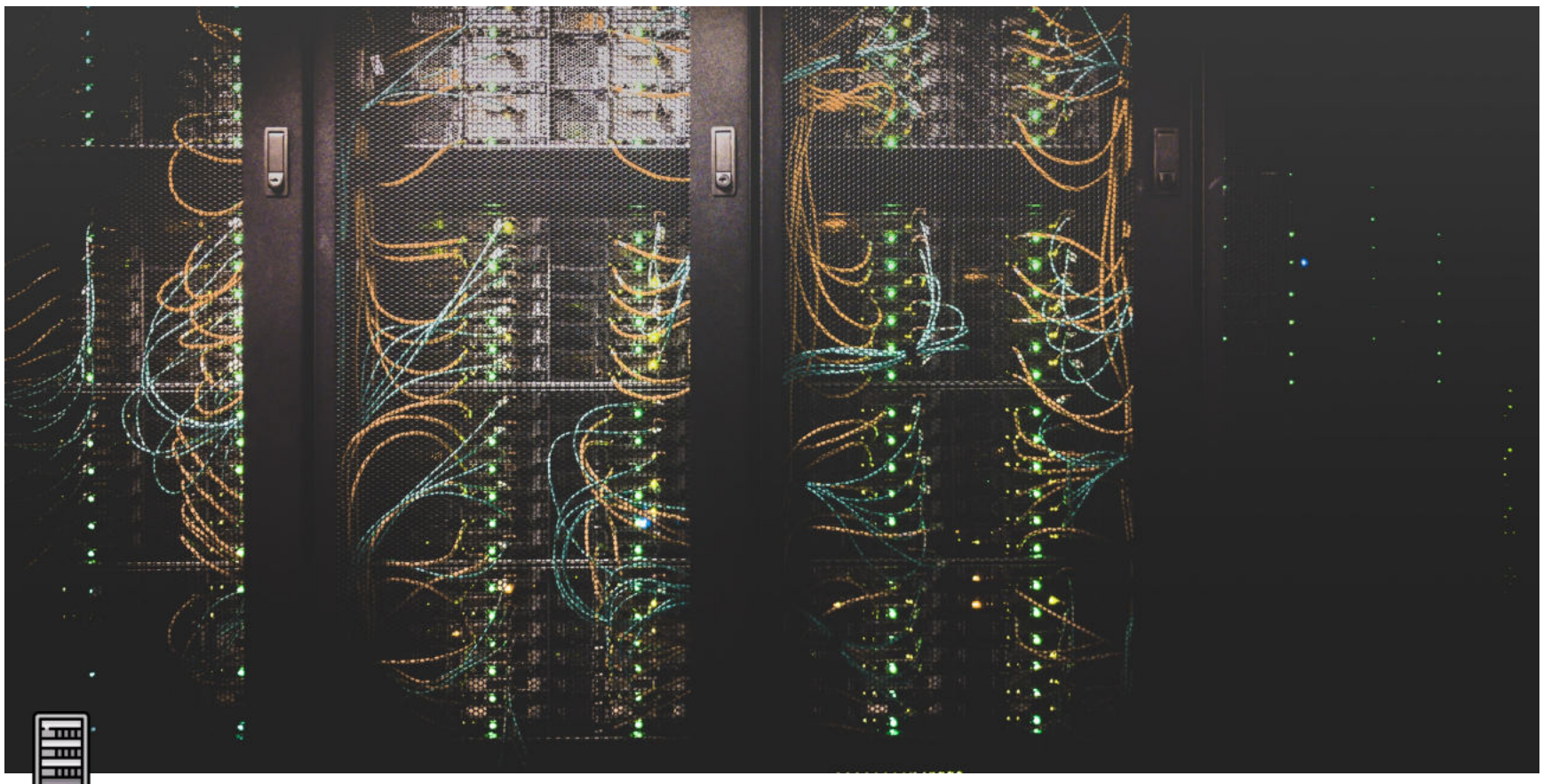


- Proceed with the installation
  - Uhh it's just Next, Next, Next .... Next ... Restart

When you install it correctly and get it up and running, you might see something like this ...



(this screenshot here is Ubuntu 21.10 and gosh that's a creepy monke)



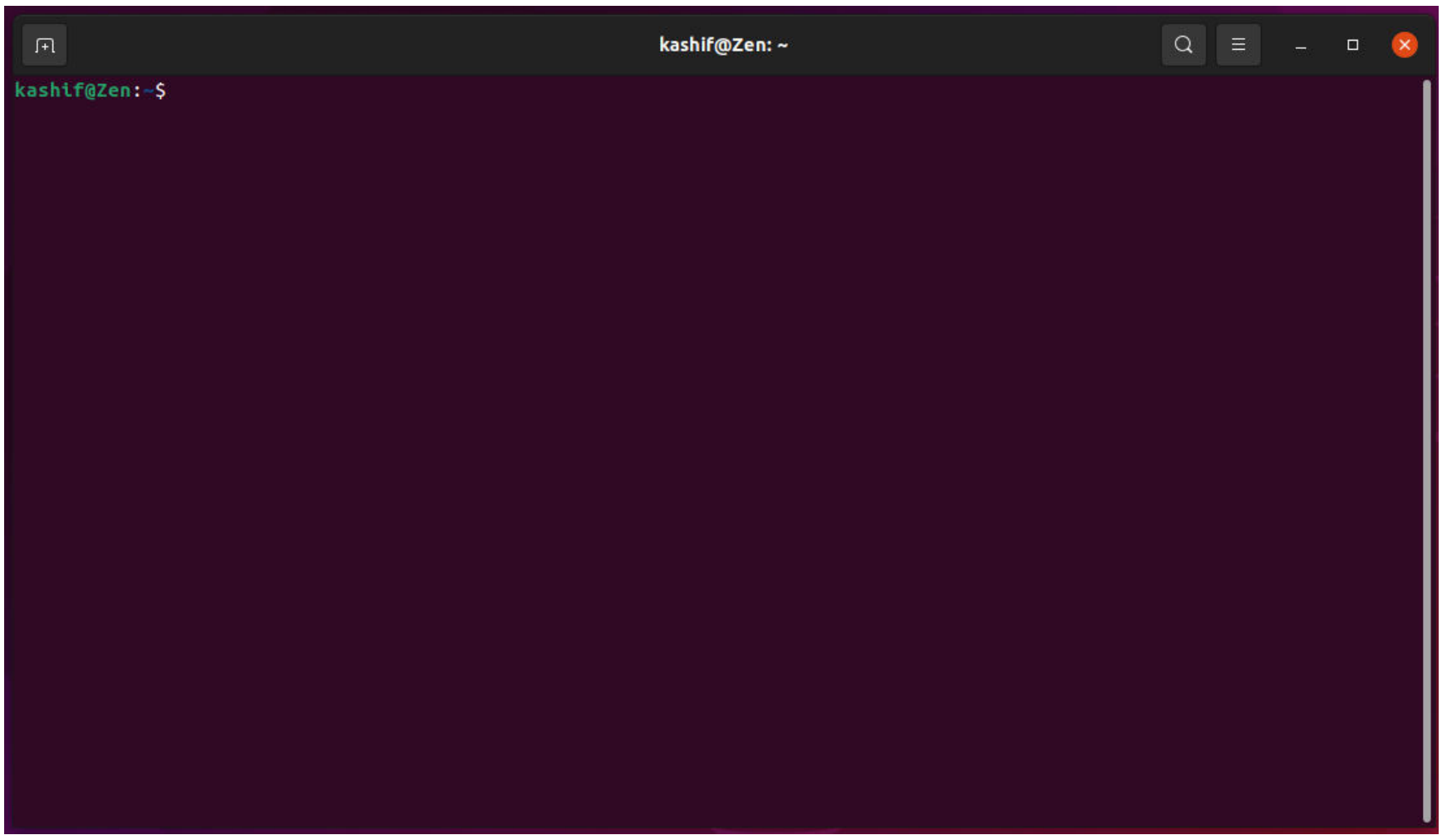
Command Line Environment

Type	📒 Lecture
Date	@December 22, 2021
Lecture #	2
Lecture URL	https://youtu.be/qrAnlpcMyYc
Notion URL	https://21f1003586.notion.site/Command-Line-Environment-f96a91e782a147a88894028d7848a2c4
Week #	1

Why use Command Line environment?

- To use linux at its max potential
- Combine commands to form powerful scripts
    - To automate using these scripts
- *To assert dominance over GUI plebs*

Terminal in Ubuntu



This is what the default terminal looks like in Ubuntu

To clear the command line

```
clear
```

- or you can press `Ctrl + L` to clear the terminal screen

To check which directory we currently are in

```
pwd
```

By default, you are placed in the home directory of the currently logged in user

To list all the files and folders in the current directory

```
ls
```

To view the currently running processes

```
ps
```

To know the OS, duh

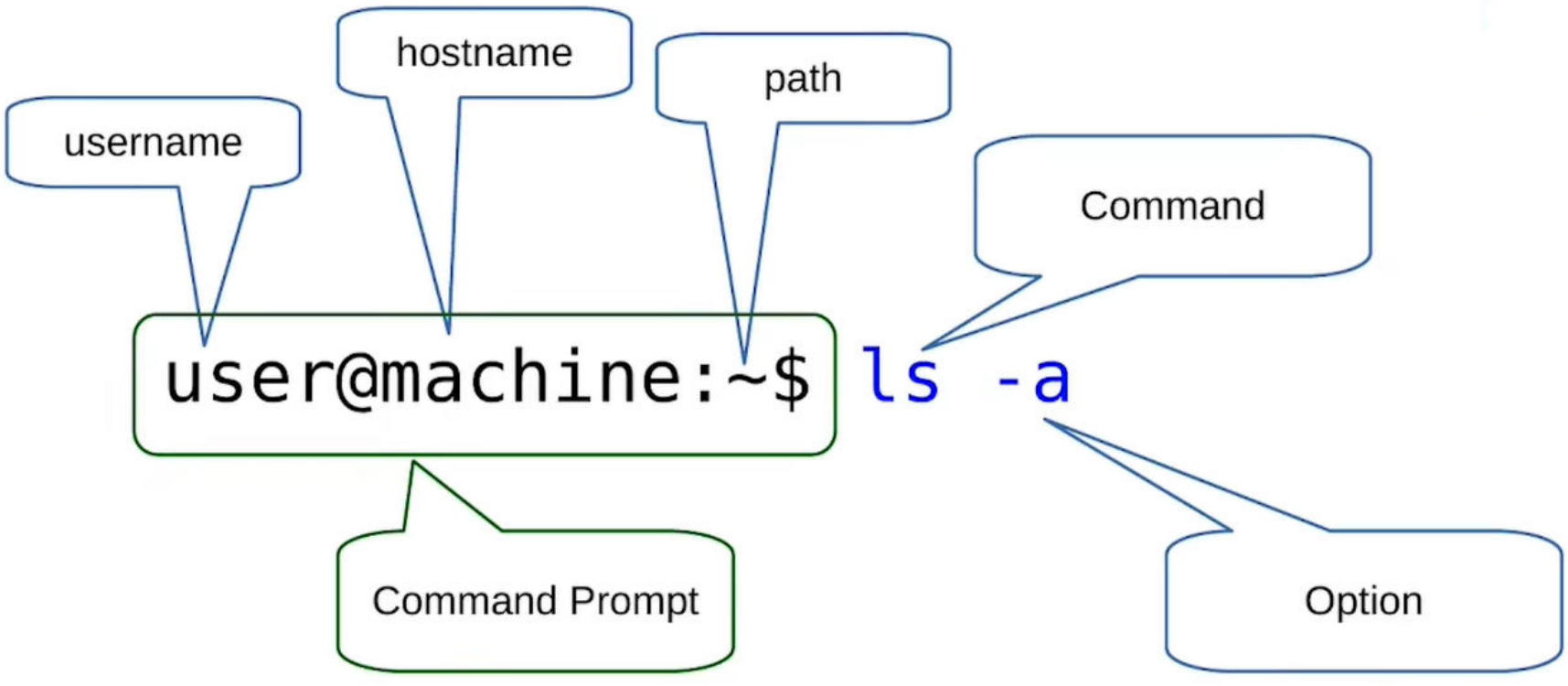
```
uname
```

To exit the shell

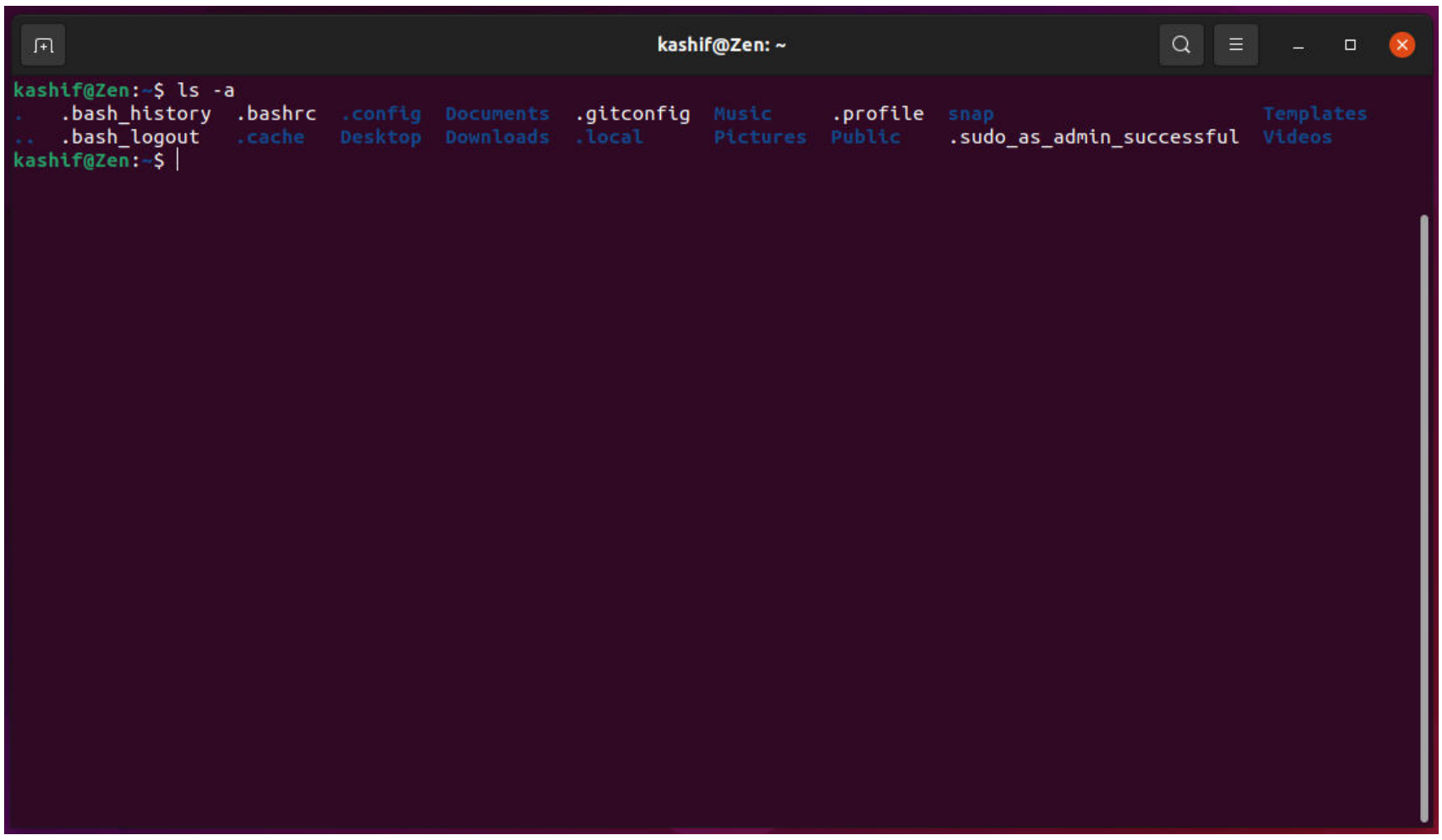
```
exit
```

- Or you can also press `Ctrl + D` to exit out of the terminal session

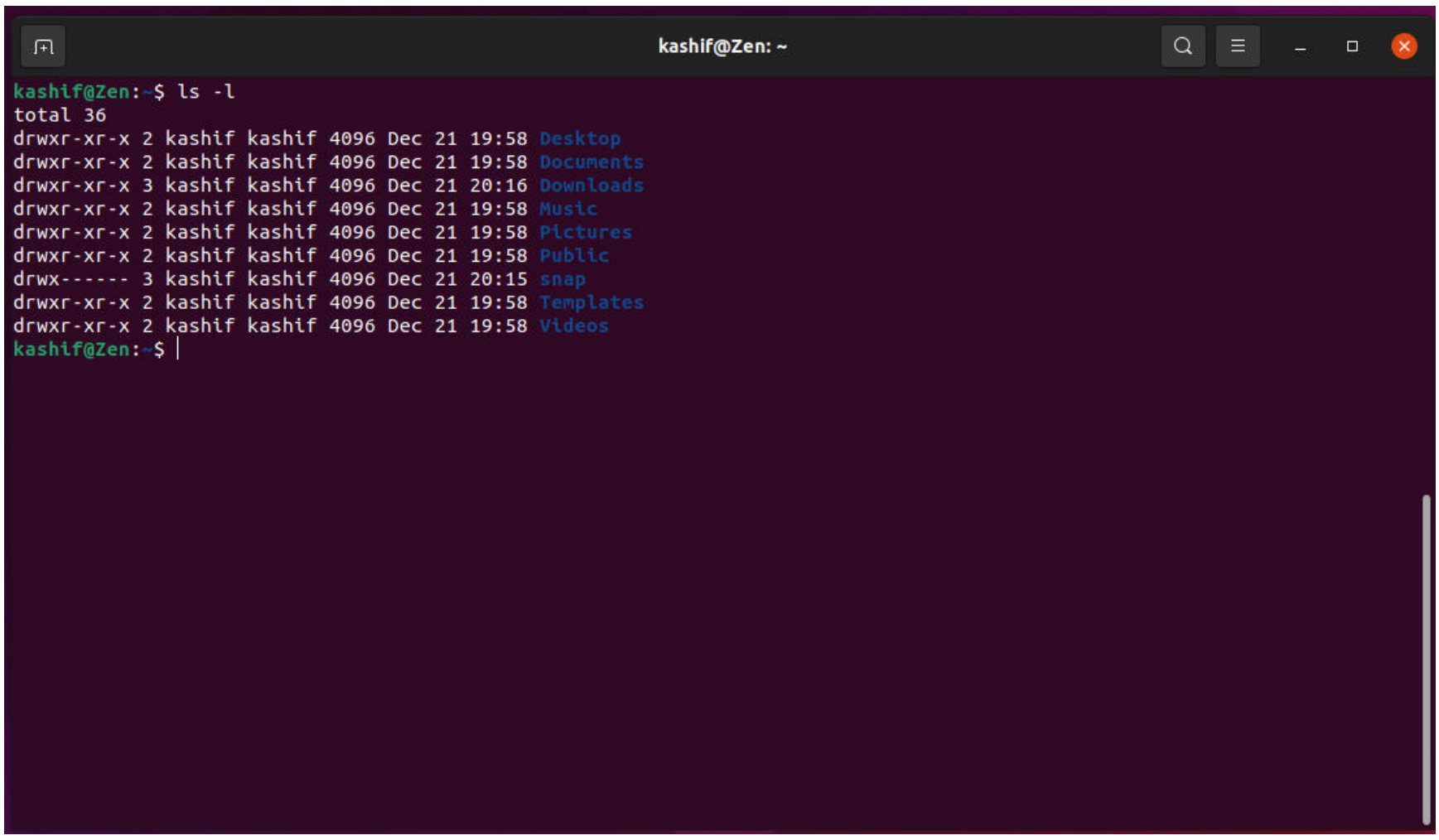
Anatomy of a typical command on the terminal



- To display all the files, press `ls -a`



- To display the files in a list, press `ls -l`



These two flags are the most commonly used ones, ***we can also combine them as one flag***

like, `ls -al`

To get help on any command

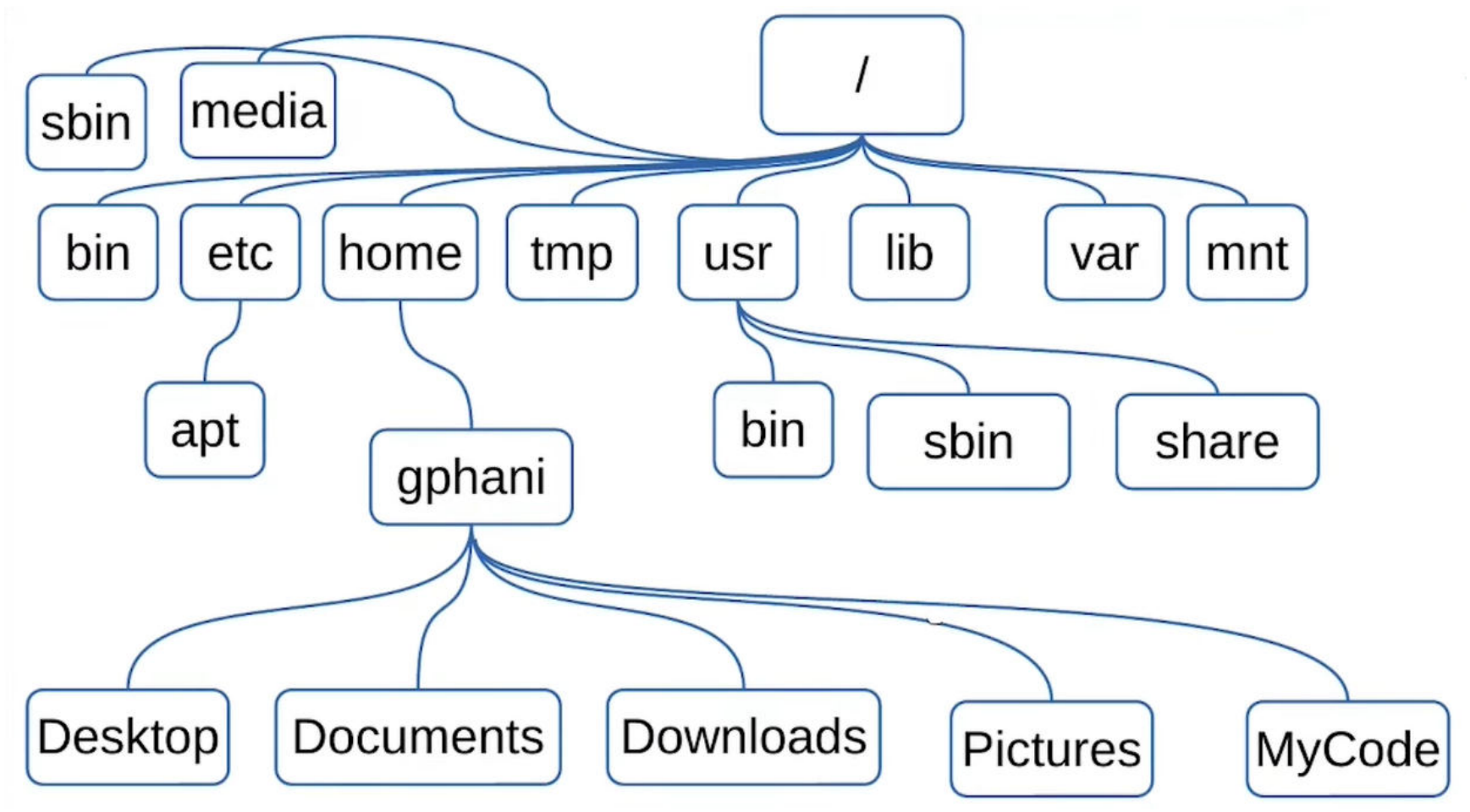
```
man <command-name>
```

- For example

  ```
  man ls
  ```

  gives us the manual for the command `ls`

Filesystem in Linux



Traversing the tree

`/` is the root of the file system

`/` is also the delimiter for sub-directories

- `.` is the current directory
- `..` is the parent directory

Path for traversal can be absolute or relative

To change directory

```
cd <location>
```

Examples

- `cd` without any arguments will take us to the home directory of the currently logged in user
- `cd <folder-name>` will change the move us into the `folder-name`
- `cd ..` will takes us to the parent of the current directory
- `cd /` will takes us to the root directory

What does this *directory* do?

- `/bin` → Essential command binaries
- `/boot` → Static files for the bootloader
- `/dev` → Device files
- `/etc` → Host specific system configuration
- `/lib` → Essential shared libraries and kernel modules
- `/media` → Mount points for removable devices
- `/mnt` → Mount points
- `/opt` → Add on application software packages
- `/run` → Data relevant to running processes
- `/sbin` → Essential system binaries
- `/srv` → Data for services
- `/tmp` → Temporary files
- `/usr` → Secondary hierarchy
- `/var` → Variable data

`/usr` hierarchy

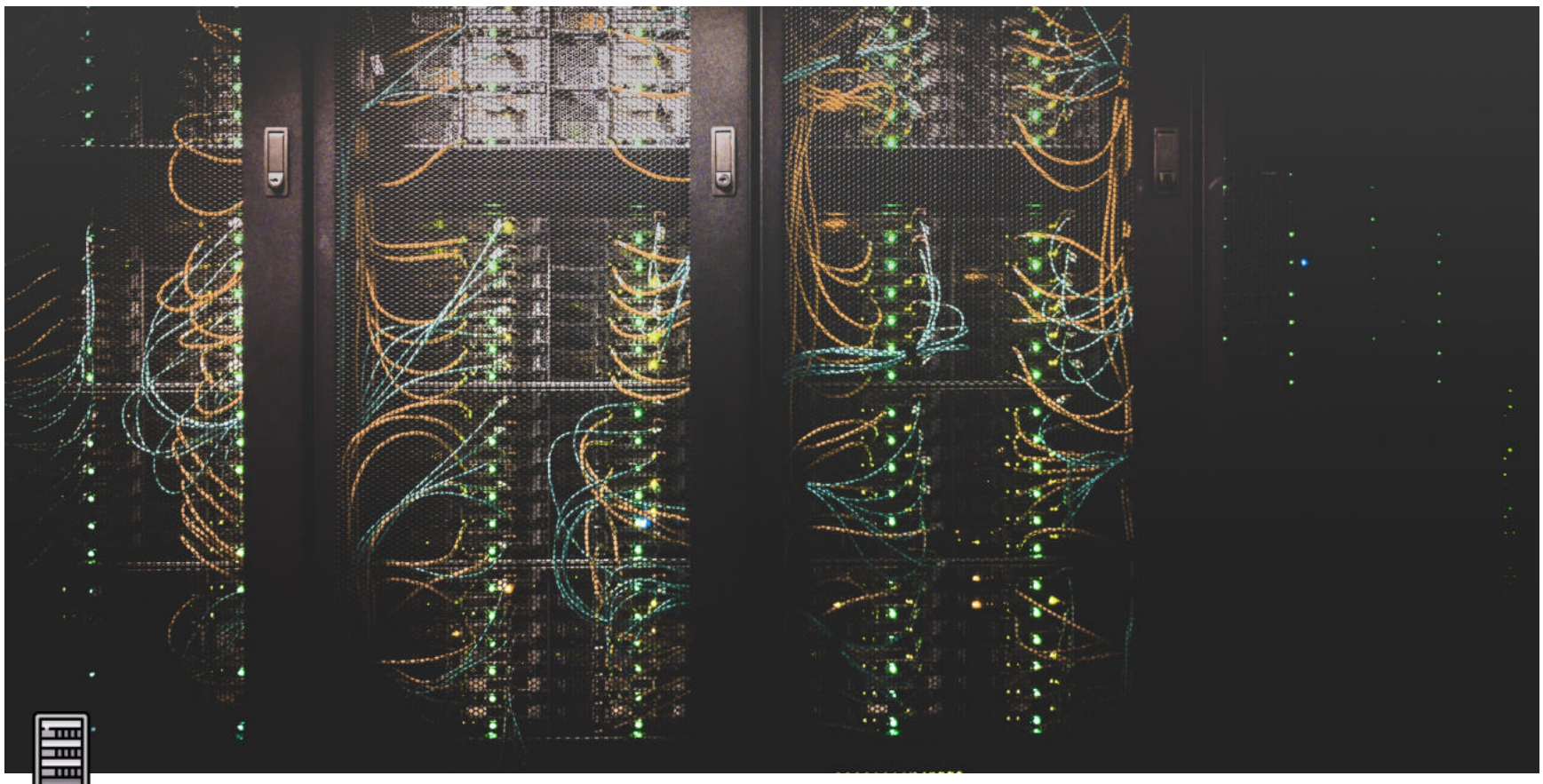
- `/usr/bin` → User commands
- `/usr/lib` → Libraries
- `/usr/local` → Local hierarchy
- `/usr/sbin` → Non-vital system binaries
- `/usr/share` → Architecture dependent data
- `/usr/include` → Header files included by C programs
- `/usr/src` → Source code

`/var` hierarchy

- `/var/cache` → Application cache data
- `/var/lib` → Variable state information
- `/var/local` → Variable data for `/usr/local`
- `/var/lock` → Lock files

- `/var/log` → Log files and directories
- `/var/run` → Data relevant to running processes
- `/var/tmp` → Temporary files preserved between reboots

	sharable	**unsharable**
static	/usr /opt	*/etc* */boot*
variable	*/var/mail*	/var/run /var/lock



Simple Commands in Linux - 1

Type	📒 Lecture
Date	@December 22, 2021
Lecture #	3
Lecture URL	https://youtu.be/DIpBEmRDHnw
Notion URL	https://21f1003586.notion.site/Simple-Commands-in-Linux-1-3e5b2c25e6d04a7499afcb89af041b20
Week #	1

Some basic commands

- `date` → Date and time

```
kashif@Zen:~/Desktop$ date
Wednesday 22 December 2021 12:05:01 PM IST
```

  - `date -R` → Gives the date in RFC5322 standard
- `cal` → Calendar of a month

```
kashif@Zen:~/Desktop$ cal
   December 2021
Su Mo Tu We Th Fr Sa
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

- `free` → Memory statistics

```
kashif@Zen:~/Desktop$ free
               total        used        free      shared  buff/cache   available
Mem:         4020444      589808     2725032       36244      705604     3170416
Swap:         945368           0      945368
```

  - `free -h` → Makes the output human readable
- `groups` → Groups to which the user belongs

```
kashif@Zen:~/Desktop$ groups
kashif adm cdrom sudo dip plugdev lpadmin lxd sambashare
```

idk what the junk is this

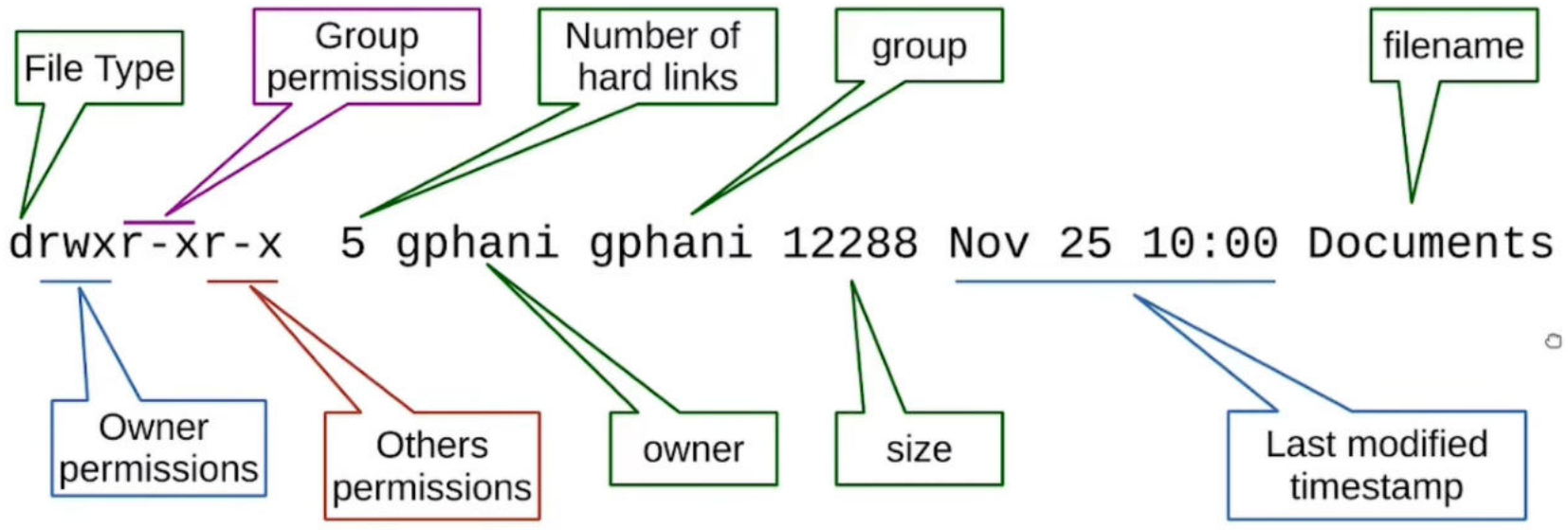
- `file` → What type of a file it is

```
kashif@Zen:~$ file .bashrc
.bashrc: ASCII text
```

- `cd -` → To visit the previous directory we were in

```
kashif@Zen:~$ cd -
/home/kashif/Desktop
kashif@Zen:~/Desktop$
```

Typical output of `ls -l`



File types

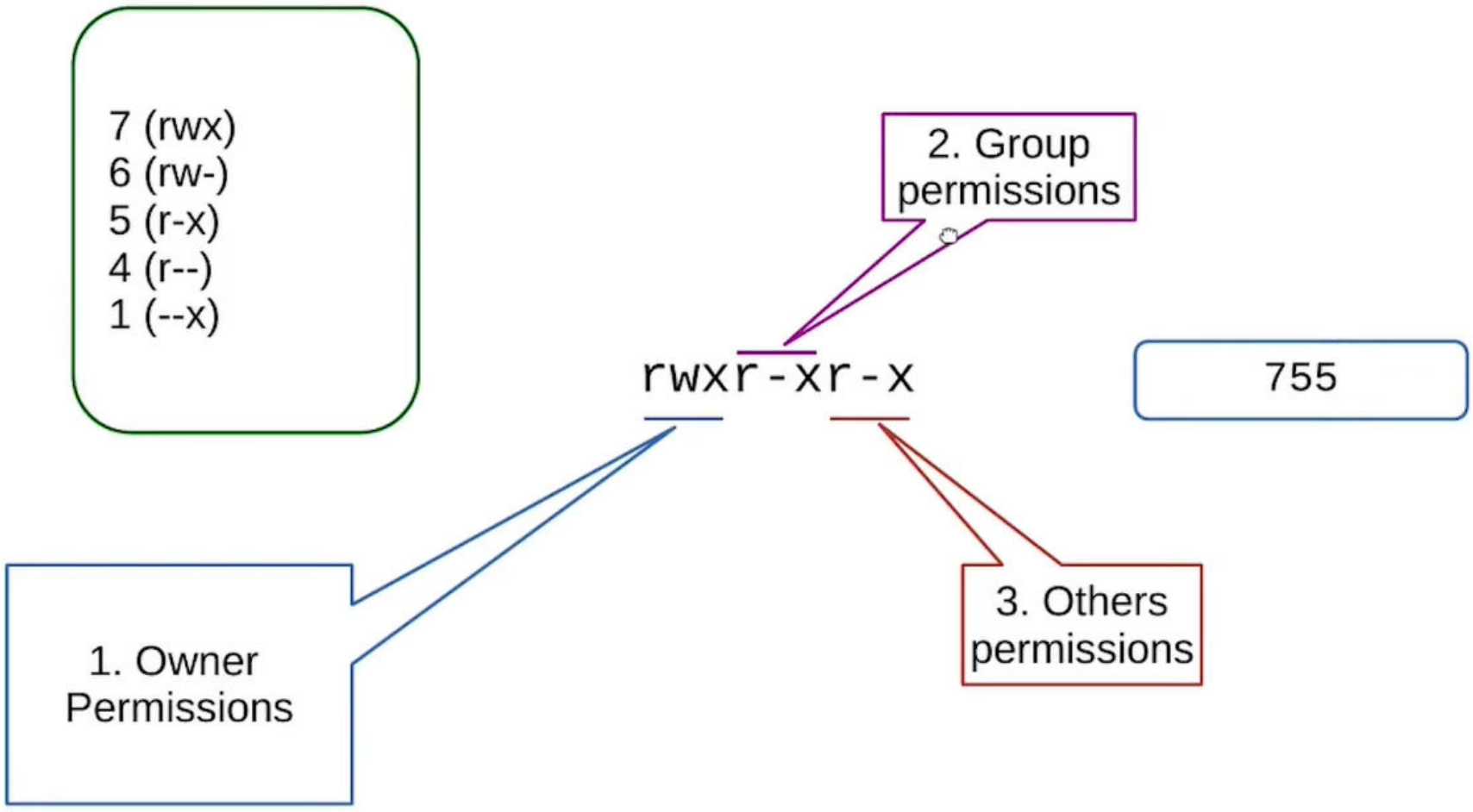
- `-` → Regular file
- `d` → Directory
- `l` → Symbolic link
- `c` → Character file
- `b` → Block file
- `s` → Socket file
- `p` → Named pipe

inode (Read: eye node)

`ls -l <name>`

- An entry in the filesystem table about the location in storage media

Permission string



To modify permissions

- Create a folder
  - `mkdir <folder-name>`
  - `chmod g-w <folder-name>` to remove the write permission from the group
  - Similarly, `chmod g-x <folder-name>` to remove the execute permission from the group
  - To add permission, `chmod g+w <folder-name>` to give write permission to the group
- So, a general structure of permission syntax is something like ...
  - `chmod <user-group><plus/minus><r/w/x> <folder-name/file-name>`
    - Where `<user-group>` are ...
      - `u` → User
      - `g` → Group
      - `o` → Others
    - `<plus/minus>` are ...
      - `-` → To remove permission
      - `+` → To add permission
    - `<r/w/x>` are ...
      - `r` → Read
      - `w` → Write
      - `x` → Execute
- We can also use numerical values for permissions
  - `chmod 700 <folder-name>` to give the `rwx` permission to user only

`touch` command

- Used to modify the timestamp of a file or folder
  - If a file does not exist, it will be created
- `touch <file-name>` to create a new file
  - `chmod 700 <file-name>` to give `rwx` permission to user only

`cp` command

- `cp <file-name> <new-name>` to copy a file to a new name
  - `cp <file-name> <new-path>` can be used to copy a file to a new path

`mv` command

- `mv <file-name> <new-path>` to move a file to a new path
  - `mv <file-name> <new-file-name>` can be used to rename a file

Also, use quotation marks if the file name includes a space

`rm` command

- `rm <file-name>` to remove a file
  - **IT WILL NOT ASK FOR YOUR CONFIRMATION**
    - Just straight up delete

    - This is the default behaviour
  - We can pass `-i` flag for the confirm remove prompt

Alias

- We can also set an alias for long commands, for example ...
  - `alias ll="ls -altrhF"`

Know current user

```
whoami
```

Read a text file, page-by-page

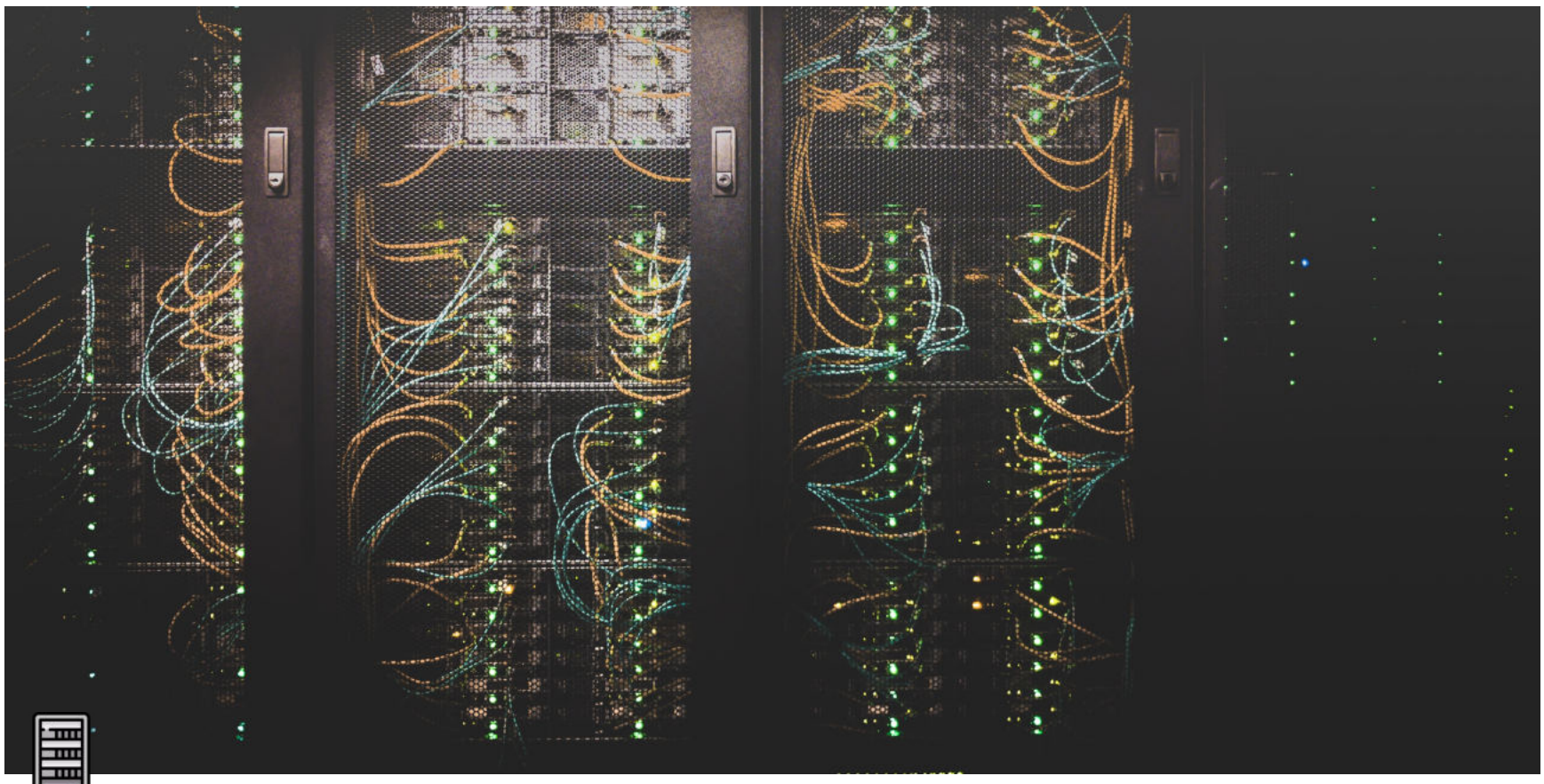
```
less <file-name>
```

To know the type of a file

```
file <file-name>
```

Some commands

- `chmod` → Change permissions of a file
- `touch` → Change modified timestamp of a file
- `cp` → Create a copy of a file
- `mv` → Rename/Move a file
- `mkdir` → Create a directory
- `rm` → Remove a file



Simple Commands in Linux - 2

Type	📒 Lecture
Date	@December 22, 2021
Lecture #	1
Lecture URL	https://youtu.be/lpe6AKk5GIk
Notion URL	https://21f1003586.notion.site/Simple-Commands-in-Linux-2-0cd2a51884c14be1a43ad24cd99c6692
Week #	2

`ls`

- Short and Long form of options
- Interpretation of directory as an argument
- Recursive listing
- Order of options on command line

  `ls -l` **will display the files & folders in the current directly, in a list**

```
kashif@Zen:~$ ls -l
total 40
drwxr-xr-x 2 kashif kashif 4096 Dec 21 19:58 Desktop
drwxr-xr-x 2 kashif kashif 4096 Dec 21 19:58 Documents
drwxr-xr-x 3 kashif kashif 4096 Dec 21 20:16 Downloads
drwxr-xr-x 2 kashif kashif 4096 Dec 22 12:35 level1
drwxr-xr-x 2 kashif kashif 4096 Dec 21 19:58 Music
drwxr-xr-x 2 kashif kashif 4096 Dec 21 19:58 Pictures
drwxr-xr-x 2 kashif kashif 4096 Dec 21 19:58 Public
drwx------ 3 kashif kashif 4096 Dec 21 20:15 snap
drwxr-xr-x 2 kashif kashif 4096 Dec 21 19:58 Templates
drwxr-xr-x 2 kashif kashif 4096 Dec 21 19:58 Videos
```

If we add the name of a folder after `ls -l`*, it will display the files & folders of that folder*

Commands to work with text files

- `less <file-name>` → To read text files, page-by-page
  - Press `q` to exit
- `cat <file-name>` → To read the entire text file, and dump them on the terminal window

    - Depending on the situation, `less` can be a better way to display the contents of a file on the terminal window rather than `cat`
- `more <file-name>` → Works similarly to `less`
    - View contents of the file page-by-page
- `head <file-name>` → Displays the first 10 lines of a file
    - Takes an optional flag, `-n` followed by an integer to display that number of lines
- `tail <file-name>` → Works the opposite of `head` command
- `wc <file-name>` → Displays the number of lines (or newlines), words and bytes in a file
    - Takes optional flags like `-l` to display only the # of lines
- `which <command>` → A command to locate the commands, *heh*

- `whatis <command>` → Gives a brief description of a command
- `apropos <keyword>` → Takes in the keyword and returns a list of commands that contain the keyword in their name or in their description

```
kashif@Zen:/etc$ apropos who
w (1)                - Show who is logged on and what they are doing.
who (1)              - show who is logged on
whoami (1)           - print effective userid
```

- `help` → Displays the reserved keywords
    - Can pass a command name as optional parameter and it'll display it's help
    - Works with `shell keyword`
- `info` → Open a file (which is similar to a webpage) that lists all the commands, and we can move the cursor to any command and press ***Enter*** to get the manual of that command
    - Use the `<` button to go back, how do you press the `<` button? `Shift + ,`
- `type` → Know the type of a command

```
kashif@Zen:/etc$ type which
which is /usr/bin/which
kashif@Zen:/etc$ type who
who is hashed (/usr/bin/who)
kashif@Zen:/etc$ type pwd
pwd is a shell builtin
kashif@Zen:/etc$ type ls
ls is aliased to `ls --color=auto'
kashif@Zen:/etc$ type type
type is a shell builtin
```

What are aliases?

Aliases are a nickname given to a command, which may want for our convenience

How to set an alias?

```
alias <name>='<your-command>'

# Example
alias ll='ls -al'
```

How to remove an alias?

```
unalias <name>

# Example
unalias ll
```

OK, How do I know all of my aliases?

```
# Type 'alias' in the bash shell
alias
```

Multiple arguments

- Second argument
- Interpretation of last argument
- Recursion is assumed for `mv` but not for `cp`

Links

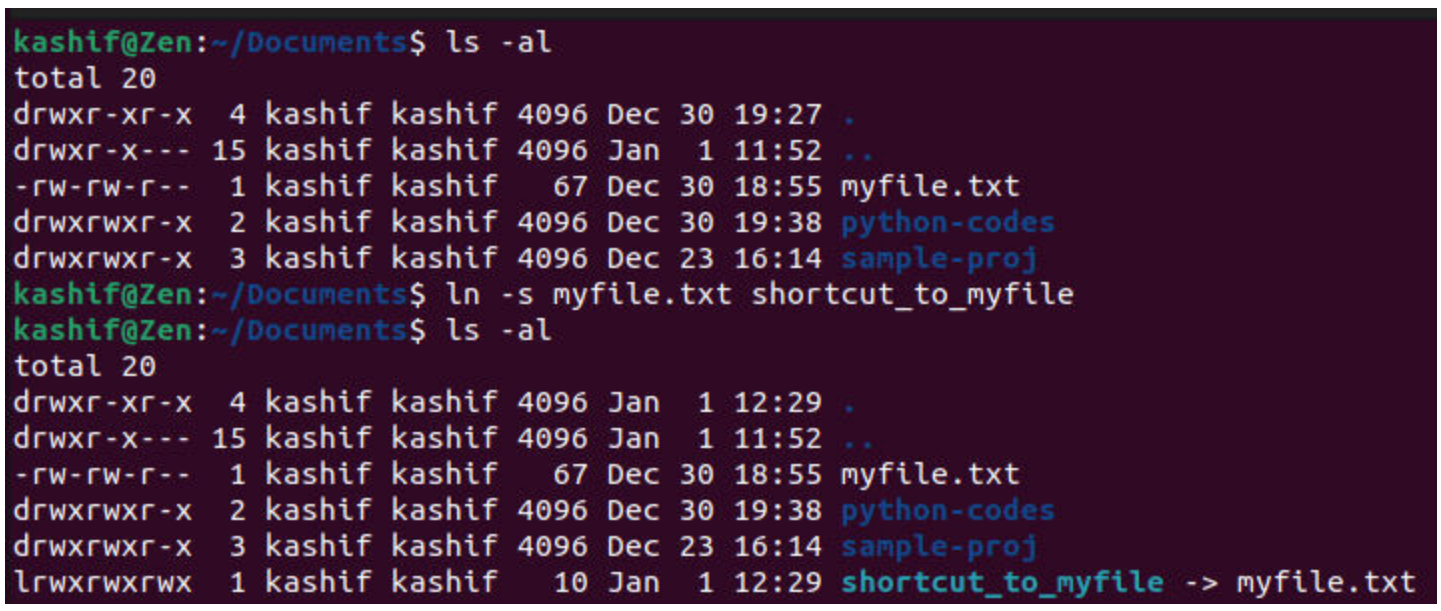
- Hard links
- Symbolic links
  - It is a bit analogous to desktop shortcuts in Windows

How to make a symbolic link?

- The command `ln` is used, with the `-s` flag

```
ln -s <source> <destination>
```

Example

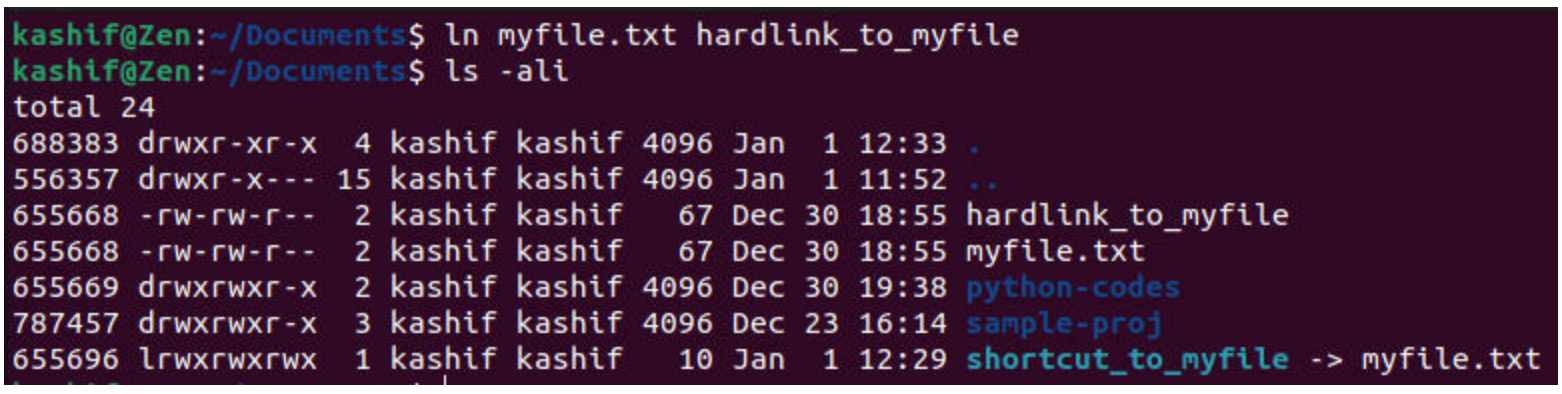


How to create a Hard link?

Just don't pass the `-s` flag to the `ln` command

```
ln <source> <destination>
```

Example



Notice the `inode` number for the hardlink is the same as the file

Size of files

- `ls -s` lists the files with file sizes

- `stat <file-name>` lists various other details as well as the file size
- `du <file-name>` shows the size of the file
  - Takes an optional `-h` flag to format the output in human readable form

In-memory filesystem

- `/proc`
- `/sys`

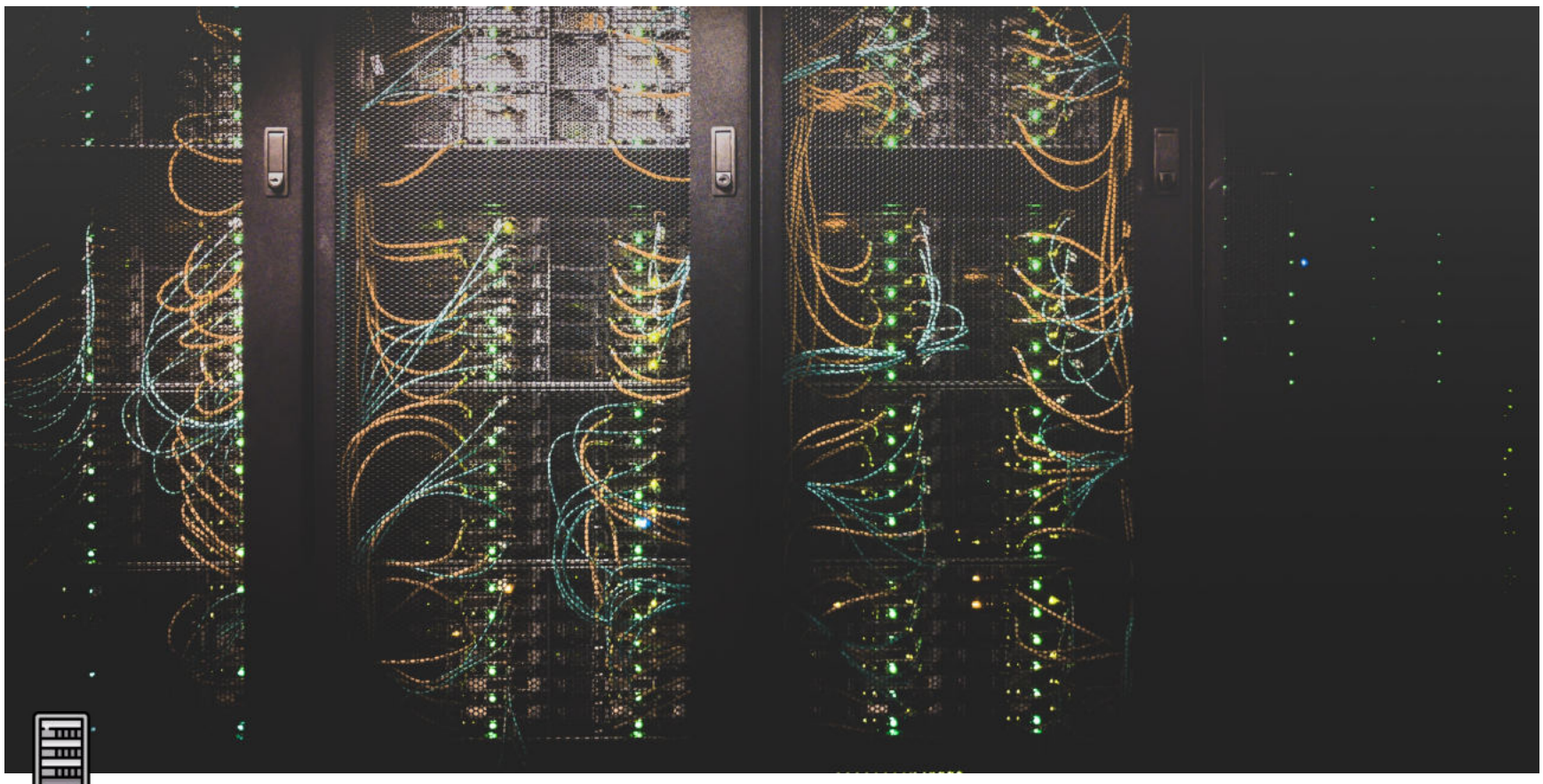
These directories are not stored on your HDD/SSD

They are located in your system memory

These contain the info about the system, so *no* `rm -rf` *-ing around as* `root`

A few commands

- `free` → To display the system memory stats, like used memory, free memory
  - Takes an optional `-h` flag to format the output in human readable form
- `df -h` → To know about the partitions, in human readable form



$hell variables

Type	📒 Lecture
Date	@December 22, 2021
Lecture #	2
Lecture URL	https://youtu.be/pPRge8Yxbso
Notion URL	https://21f1003586.notion.site/hell-variables-7c2117f90cee4aa7aec3ee410038f32a
Week #	2

`echo`

`echo` command prints a string or a environment variable

Example:

```
echo $HOME
echo Hello, World
```

Frequently used shell variables

- `$USERNAME`
- `$HOME`
- `$HOSTNAME`
- `$PWD`
- `$PATH`

Special shell variables

- `$0` → Name of the shell
- `$$` → Process ID of the shell
- `$?` → Return code of previously run program
- `$-` → Flags set in the bash shell

Process control

- Use of `&` to run a job in the background
- `fg`

- `coproc`
- `jobs`
- `top`
- `kill`

Program exit codes

- 0 → success
- 1 → failure
- 2 → misuse
- 126 → command cannot be executed
- 127 → command not found
- 130 → processes killed using `Ctrl + C`
- 137 → processes killed using `kill -9 <pid>`

Flags set in `bash`

- `h` → locate and hash commands
- `B` → brace expansion enabled
- `i` → interactive mode
- `m` → job control enabled
- `H` → ! style history substitution enabled
- `s` → commands are read from stdin
- `c` → commands are read from arguments

`echo`

```
echo Hello World
echo "How do you do?"
```

it's a start

We can enclose the string in quotes as well

Make sure to match the quotes properly

Some common `echo` commands

```
echo $USERNAME
echo $USER
echo $PWD
echo $HOME
echo $HOSTNAME
echo $PATH
```

We can enclose the shell variable (marked as $) in double quotes, and it'll replace the value

However, if we use single quote around the shell variable, the shell variable is printed as it is

We can also escape the shell variable by using a backslash in front of it

Example → `echo "User is \$USERNAME"` will display `User is $USERNAME`

To view all the shell variables

```
printenv
env
```

`printenv` prints all or part of the environment

```
printenv HOME
```

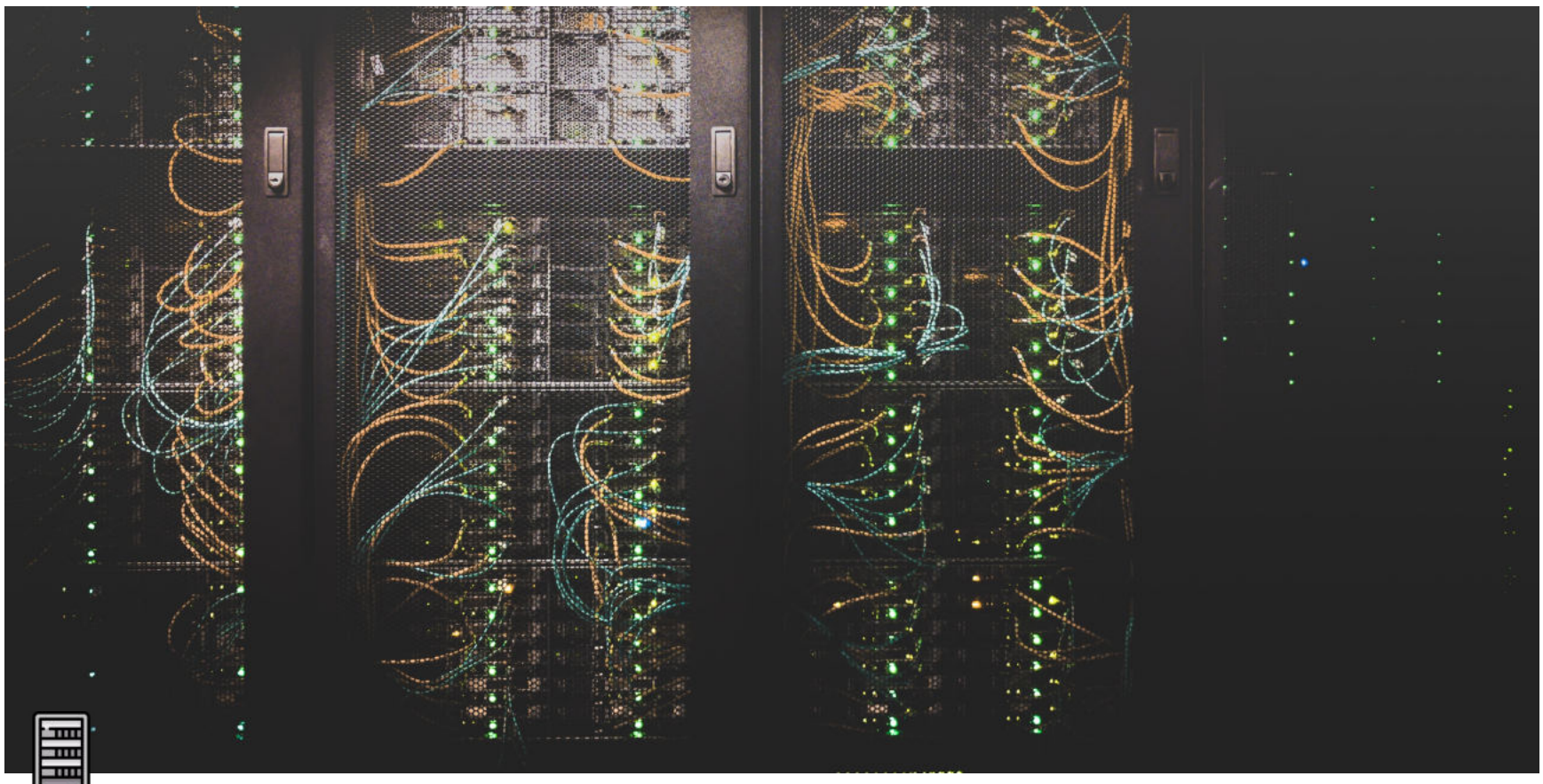
`set`

`set` allows you to change the values of shell options and set the positional parameters, or to display the names and values of shell variables. (Source)

To escape any aliases set already

```
\<alias>

# Example
\ls
\date
```



Linux Process Management

Type	📒 Lecture
Date	@December 22, 2021
Lecture #	3
Lecture URL	https://youtu.be/2aThmDRvSWU
Notion URL	https://21f1003586.notion.site/Linux-Process-Management-d73af14458004045b281f5995be7cf32
Week #	2

`sleep` command

- It is a delay for a specified amount of time
  - The time is specified in seconds

Example

```
# The following command will make the prompt sleep for 5 seconds
sleep 5
```

Coprocess

- The shell command → `coproc`
- A Coprocess is executed asynchronously in a subshell
  - In simple words, this command is used to run a process without actually losing control of the prompt
  - As we don’t lose prompt access, we can execute other commands

Usage

```
coproc sleep 20
```

When the above mentioned process is completed, the prompt will output a message saying `Done`

We can run the `ps` command to view the running status of the aforementioned command

`kill`

- This command is used to kill an already running process

- `Ctrl + C` *is also quite handy*

Usage

```
kill -9 <PID>
```

How do I know the PID?

`ps` command, try `ps --forest` instead

`&` sign to put a process in the background

- We can also put the `&` (ampersand) sign at the end of a command to put that process in the background

Usage

```
sleep 30 &
```

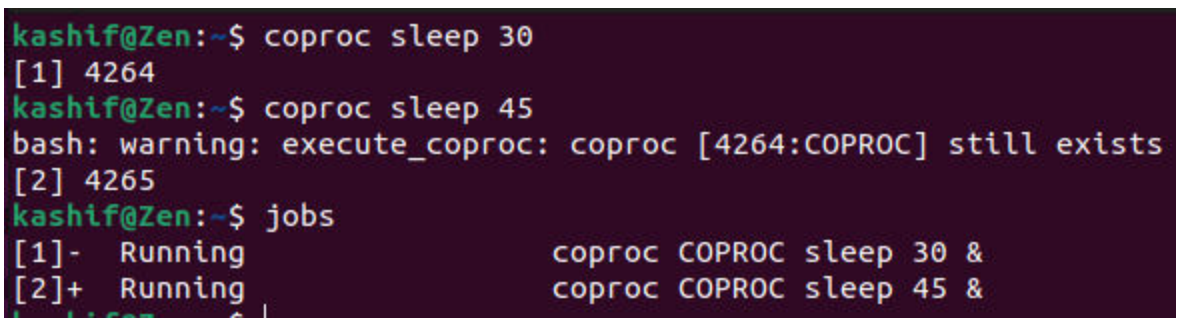
Alright, how do I bring it to the foreground?

```
fg
```

`jobs`

List the commands that are running in the background

Example



`top` command

It's like your Windows Task Manager, but in the Linux terminal

To exit out of `top`*, press* `Q` *or* `Ctrl + C`

Pressing `Ctrl + Z` while `top` is open suspends the process, then you can do your work and come back to it by using the `fg` command

`jobs` command will show the top process

Well, `Ctrl + Z` ***will suspend any running process***

$-

```
echo $-
```

The output of this shell variable would tell us about the bash shell we are currently using

To know what the output means, type `man bash` and match the flags

Launch a bash shell which is not interactive

```
bash -c "echo \$-"
```

```
bash -c "echo \$-; ps --forest;"
```

To know the PID of the new bash shell we launcher

```
bash -c "echo \$$; echo \$-; ps --forest;"
```

`history`

The following command displays out all the commands that have been run on the current shell, chronologically

```
history
```

The following command will run the `n`-th command that has been run, make sure to put `n` from the list of commands output from the above command

`n` is integer

```
!n
```

The following command will run the previous command that was executed on the bash shell

```
!!
```

Brace Expansion

So the output of `echo $-` had a `B` flag, it refers to Brace Expansion

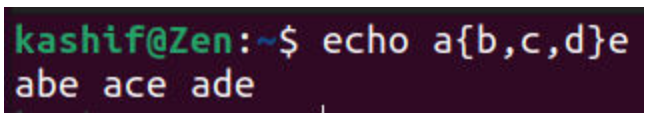
is this a JJK reference? ブレース展開

So, What is a Brace Expansion?

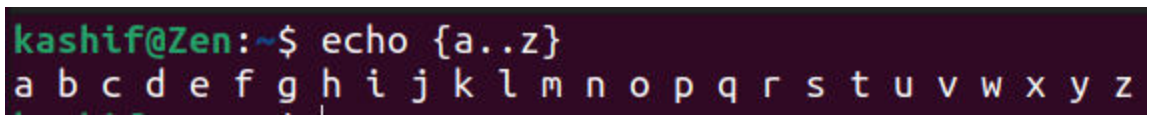
- Brace expansion is a mechanism by which arbitrary strings may be generated. Source

Example

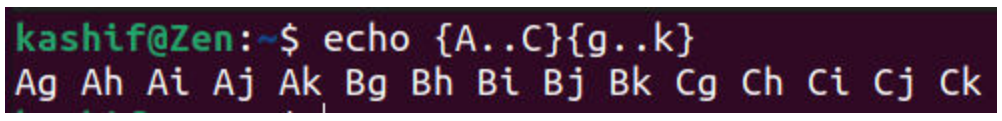
```
echo a{b,c,d}e
```




```
echo {a..z}
```




```
echo {A..C}{g..k}
```




```
# The * is a wildcard character, it expands to all the files in the current dir
echo *
```

Similarly ...

```
# The following command will return all the files folders that start with D
echo D*
```

Exit codes

- `0` → All good, no error

- `1` → General errors, misc. errors
- `2` → Misuse of shell built-in
- `126` → Command invoked cannot execute
- `127` → Command not found
- `128` → Invalid argument to exit
  - `exit` takes int from `0 - 255`
- `128+n` → Fatal error signal "n"
  - When a process running in another place (like in another shell) was killed from somewhere else (killed from not the same shell)
  - `kill -9 $PPID` → `$?` returns 137 $(127 + 9)$
- `130` → Script terminated by `Ctrl + C`
- `255*` → Exit status out of range

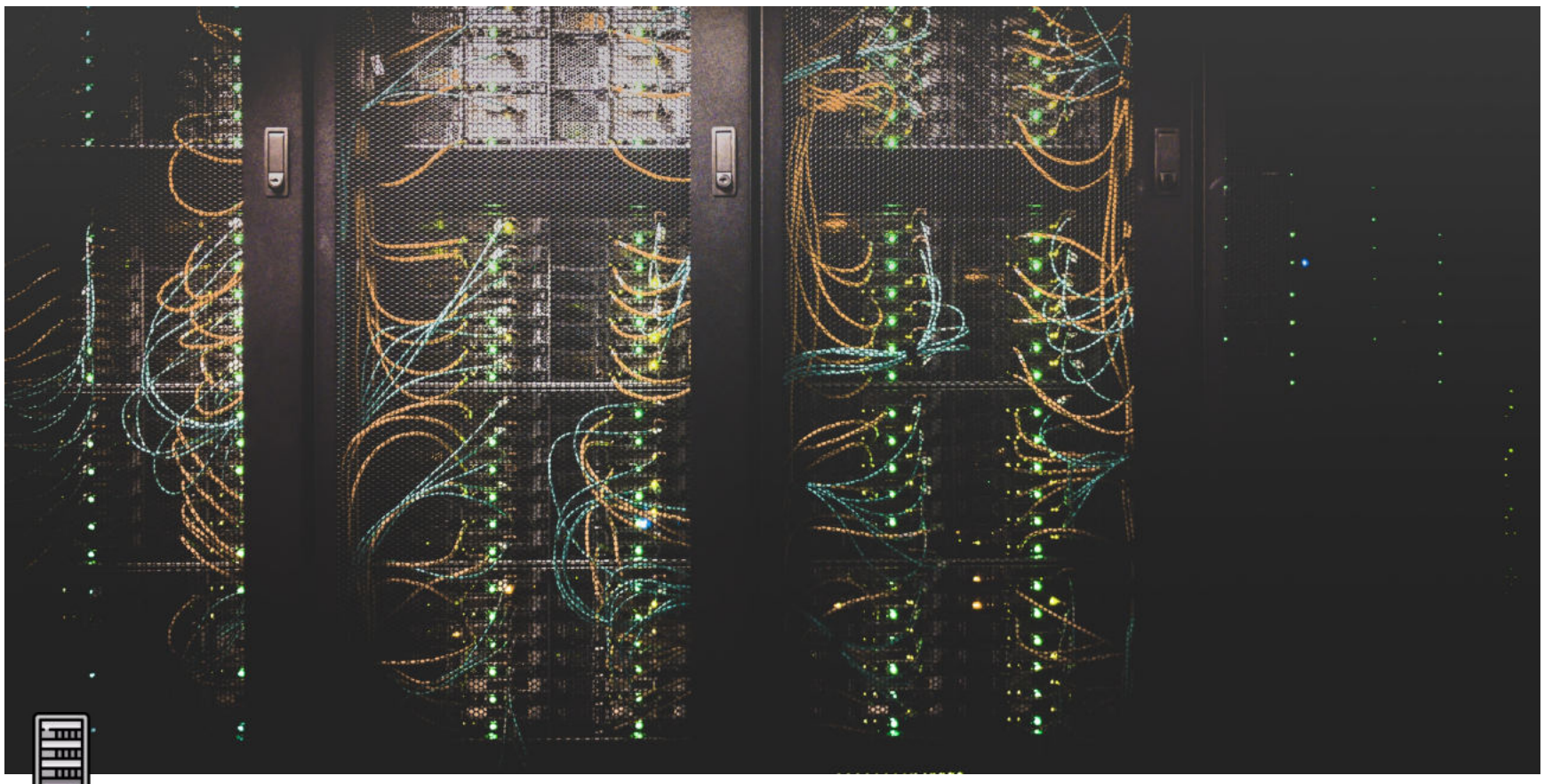
If the exit code is out of range, bash takes the modulo of the exit code w.r.t. 256

Example

```
# Create a new bash subshell and exit with the status code 300
bash -c "exit 300"

# Check the exit code
echo $?

# The output will be ...
44
```



Combining commands and files

Type	📒 Lecture
Date	@January 9, 2022
Lecture #	1
Lecture URL	https://youtu.be/Lcx9UsS7y8Y
Notion URL	https://21f1003586.notion.site/Combining-commands-and-files-2476c1091a704743840ae8b76ab078c9
Week #	3

Executing multiple commands

- `command1; command2; command3`
  - Each command will be executed one after the other
- `command1 && command2 && command3`
  - This works as a logical AND
  - The subsequent commands after `command-n` will not run if the previous command resulted in an error
- `command1 || command2 || command3`
  - This works as a logical OR
  - The subsequent commands after `command-n` will not run if the previous command resulted in a success

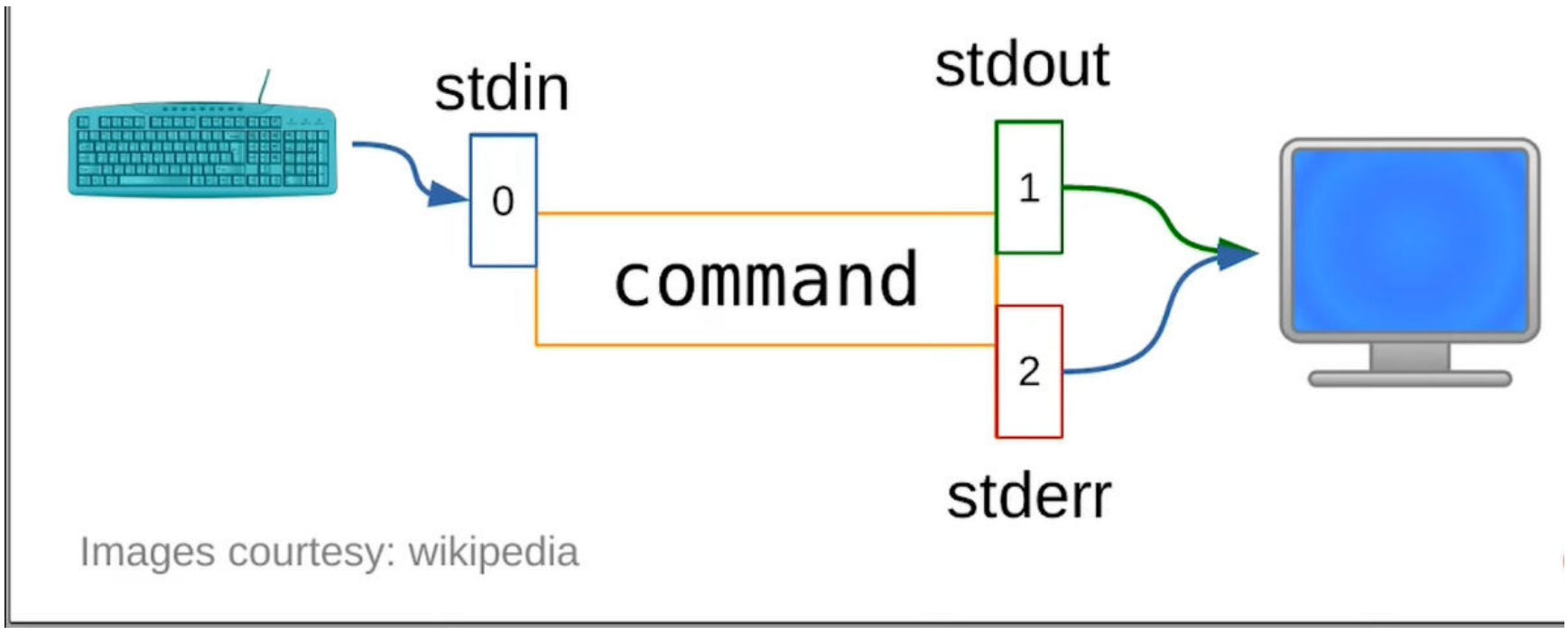
`(<command>)`

We can run any command enclosed within parentheses to execute them in a subshell, and returned back the result

We can execute a subshell within a subshell too

```
kashif@Zen:~$ echo $BASH_SUBSHELL
0
kashif@Zen:~$ (echo $BASH_SUBSHELL)
1
kashif@Zen:~$ (echo $BASH_SUBSHELL; (echo $BASH_SUBSHELL))
1
2
```

File descriptors

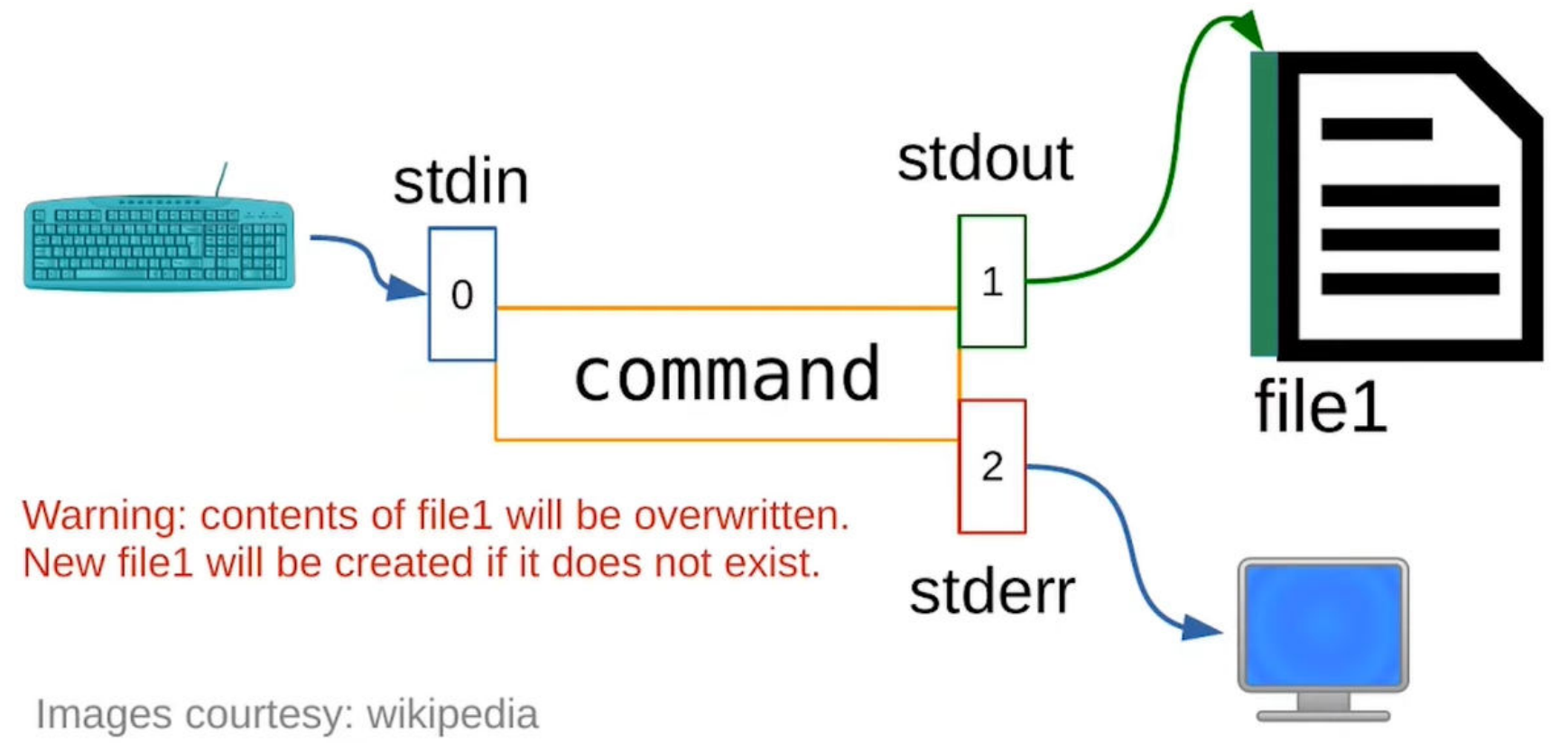


Every command in Linux has 3 file descriptors

- `stdin` (0)
  - It is a pointer to a stream that is coming from the keyboard (or the user input)
- `stdout` (1)
  - Points to the screen where the output is made
- `stderr` (2)
  - Points to the screen where the output is made

`command > file1`

- The output of the `command` should be written to `file1`



Create a file using `cat` command

`cat > filename`

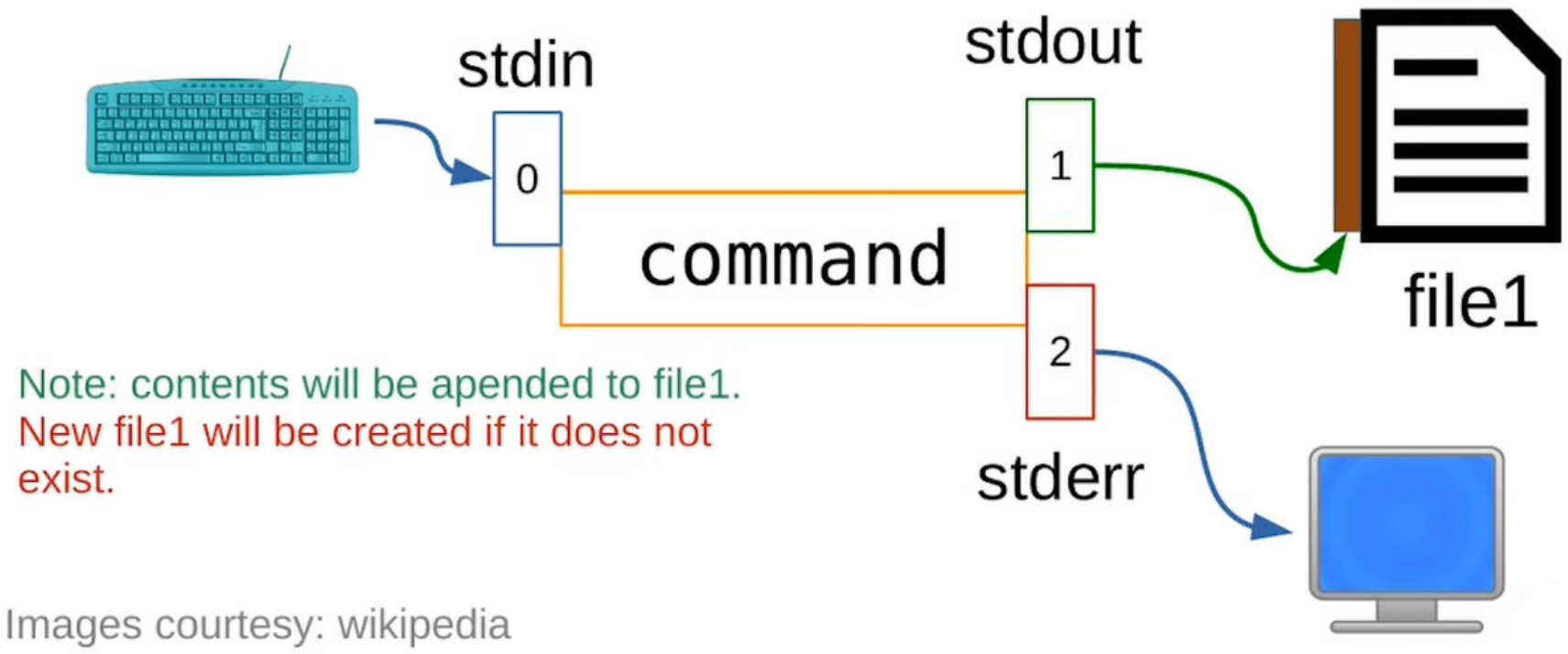
When we type this command, the `cat` command is supposed to receive the input from a file that is listed in the command line, but instead, we left that intentionally blank

So, the `cat` command, instead, reads the content from the `stdin`, i.e. the keyboard

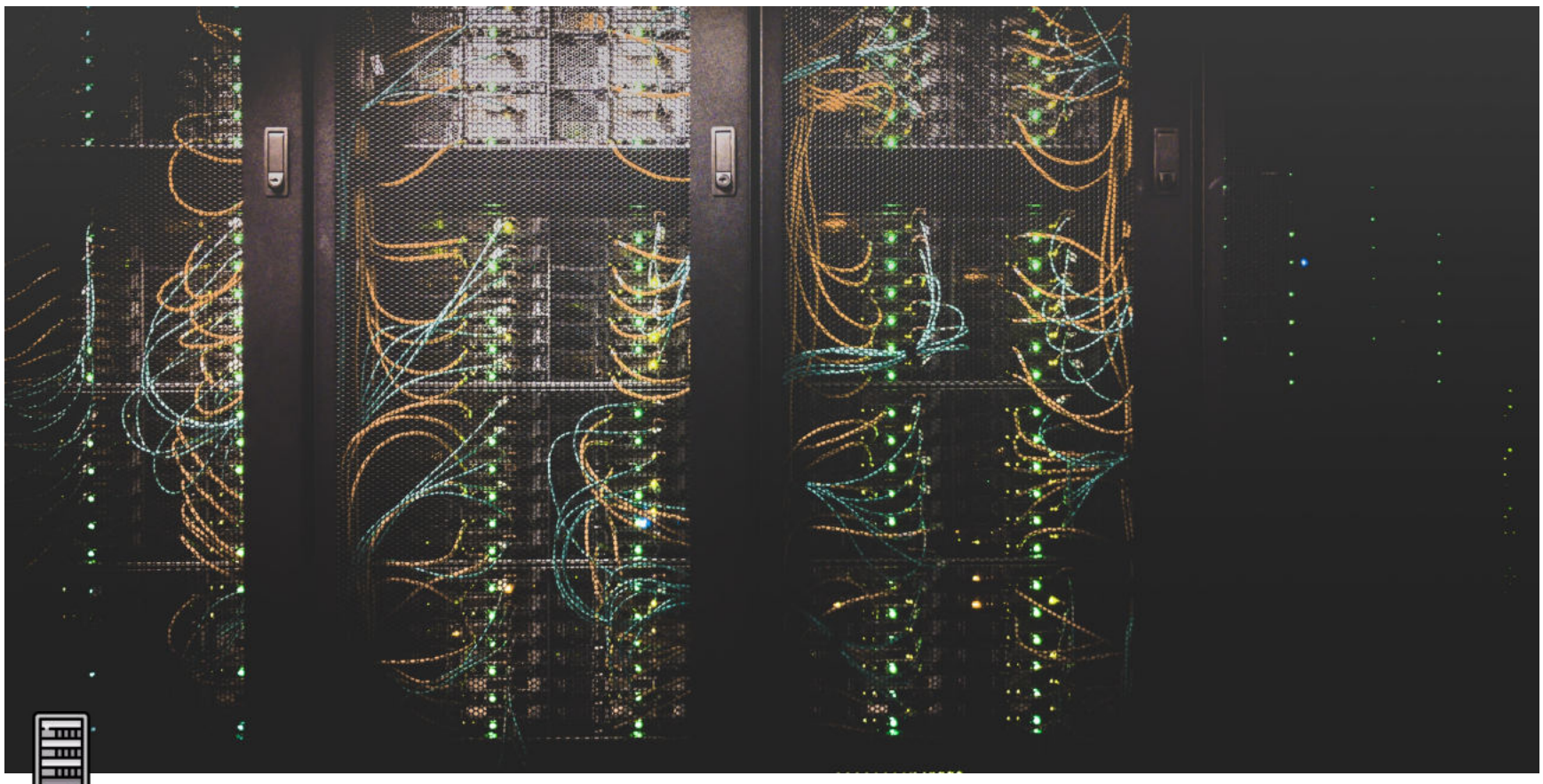
To exit, press `Ctrl + D`

`command >> file1`

- The output of `command` will be appended to `file1`



Similarly, we can use `>>` instead of `>` while creating a new file using the `cat` command

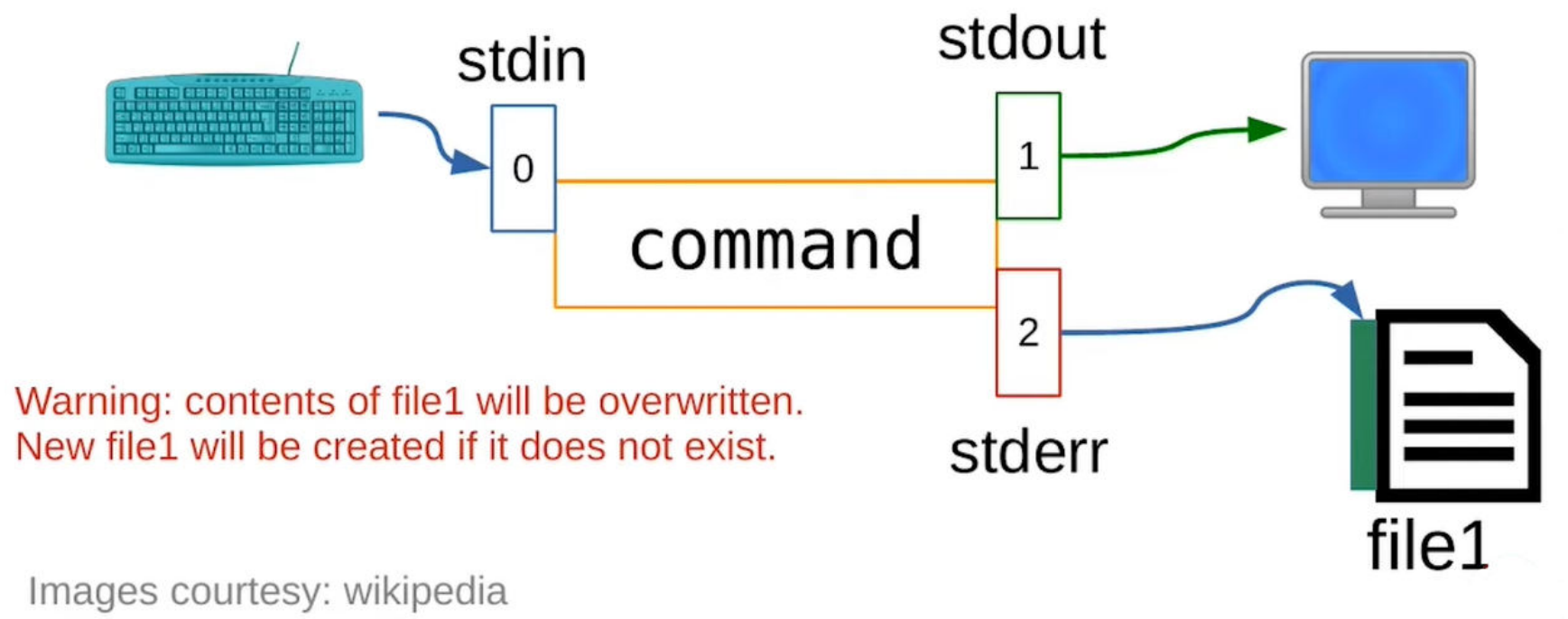


Redirections

Type	📒 Lecture
Date	@January 9, 2022
Lecture #	2
Lecture URL	https://youtu.be/BBh69kH_G_Y
Notion URL	https://21f1003586.notion.site/Redirections-734673f36f21448f99de25ccb092c8d4
Week #	3

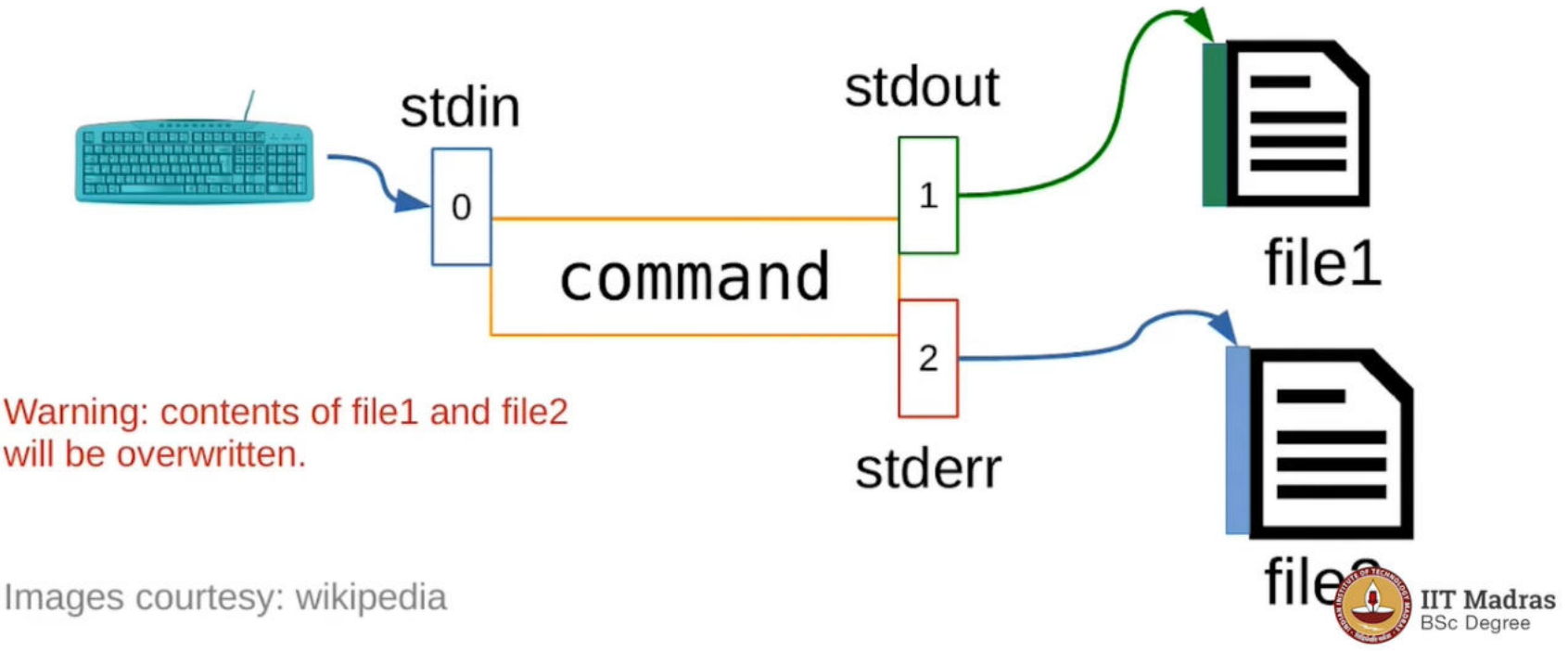
`command 2> file1`

- Redirect the output of the `command` to `stdout`, which is the display in this case
- Redirect the error of the `command` to `file1`



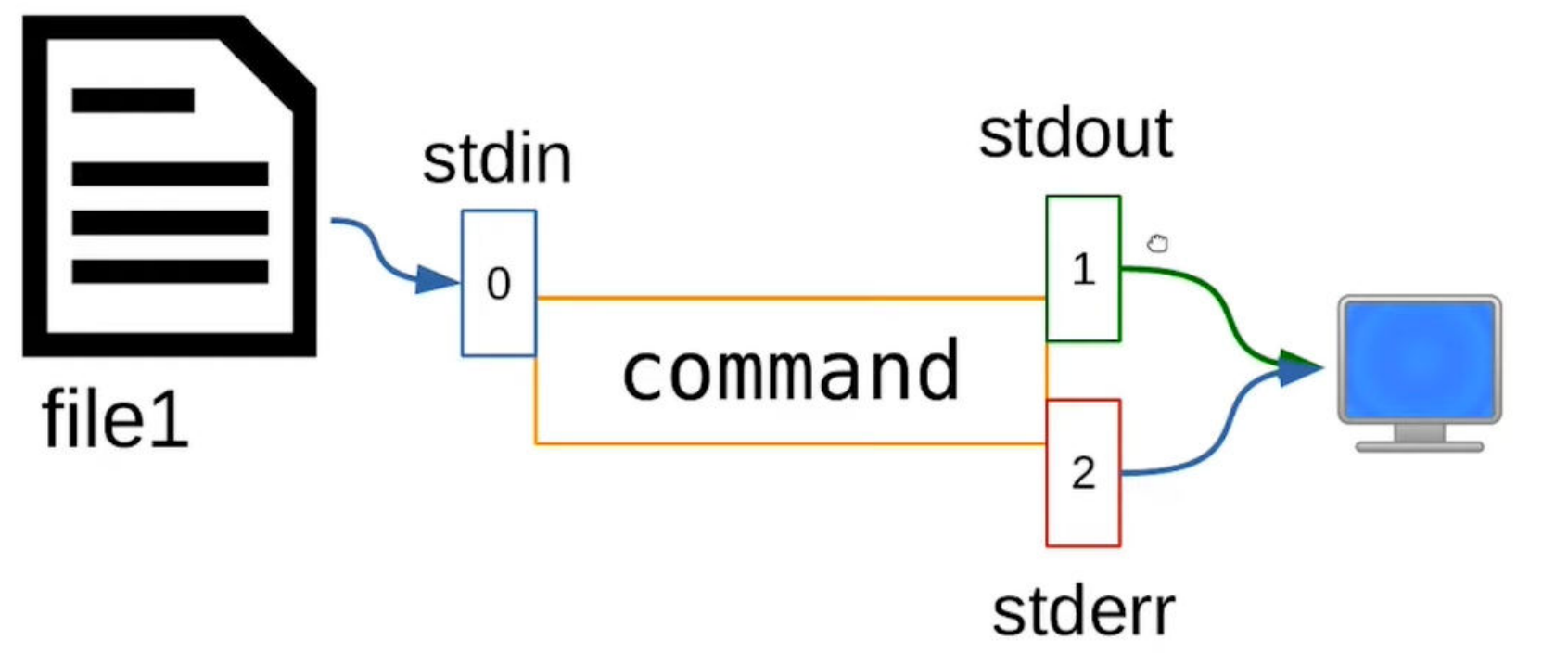
`command > file1 2> file2`

- Redirect the output of the `command` to the `stdout`, i.e. `file1`
- Redirect the error of the `command` to `stderr`, i.e. `file2`



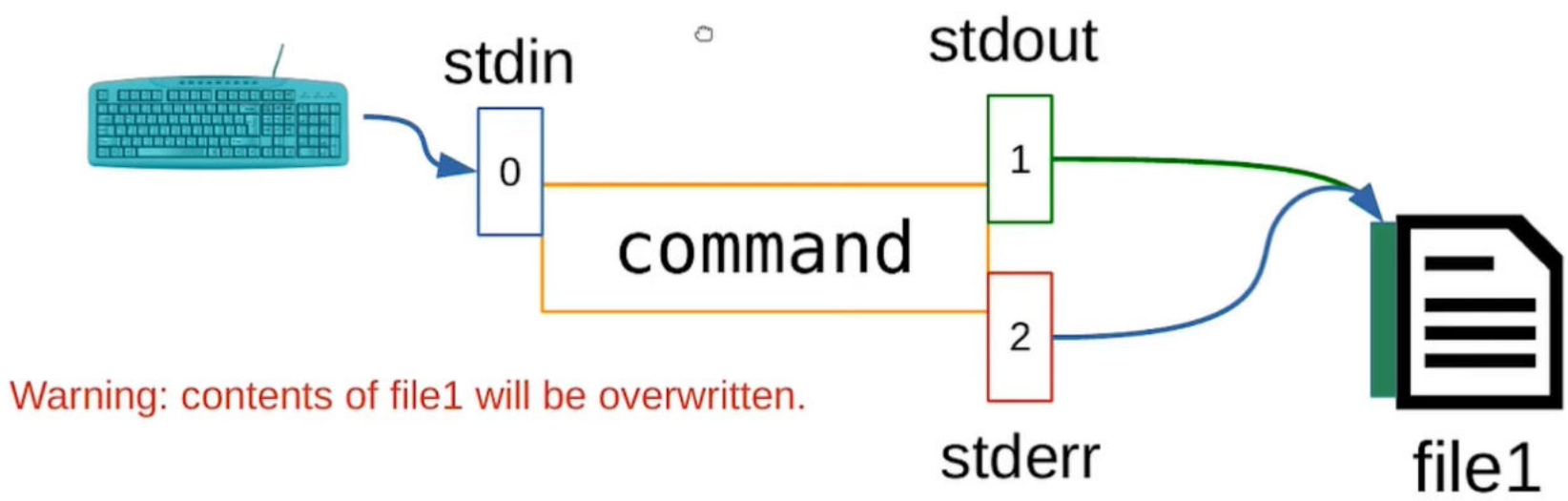
`command < file1`

- Any `command` which takes input from the keyboard, now takes input from `file1`



`command > file1 2>&1`

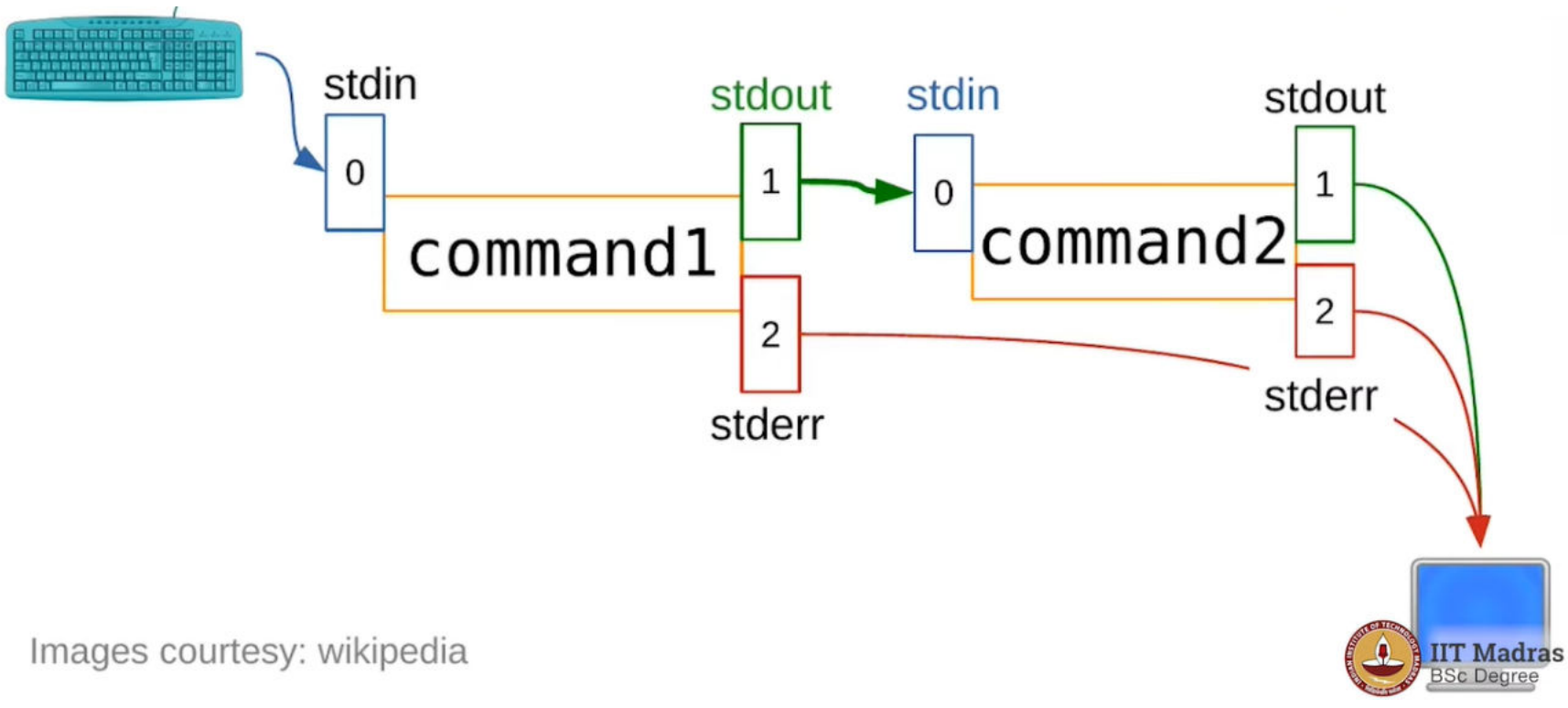
- The output of `command` is written to `file1`
- The error is redirected to stream 1, which is `stdout`



`command1 | command2` → **pipe operator** `|`

- The output of `command1` is sent to `command2` as input

- By default, the `stderr` will output to the display



Count the number of files in the directory `/usr/bin`

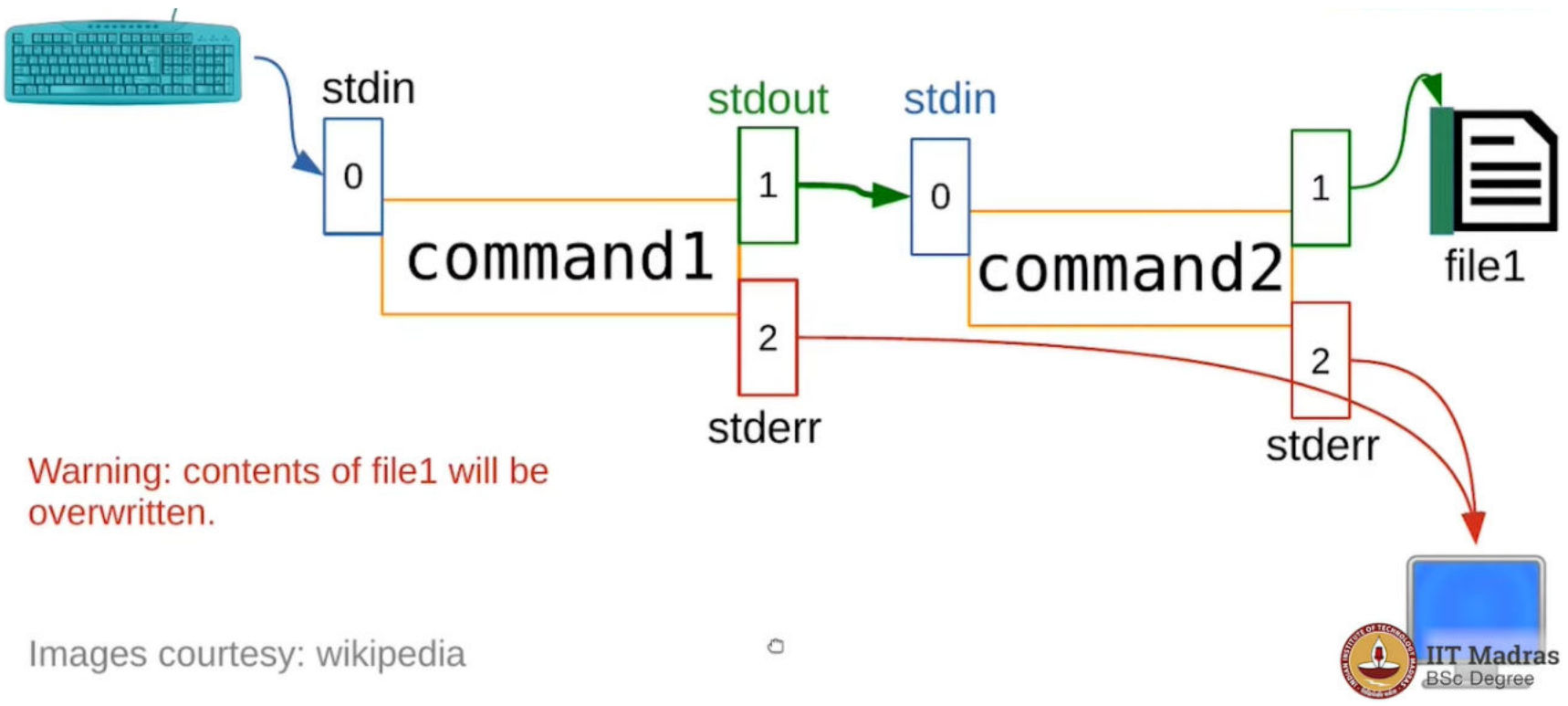
```
kashif@Zen:~$ ls /usr/bin/ | wc -l
1603
```

List the files of `/usr/bin` directory, but use the `less` command to scroll at ease

`ls /usr/bin | less`

`command1 | command2 > file1`

- The `stdout` of `command1` is mapped to `stdin` of `command2`
- The `stdout` of `command2` is written to `file1`
- The `stderr` is output to the display



`/dev/null`

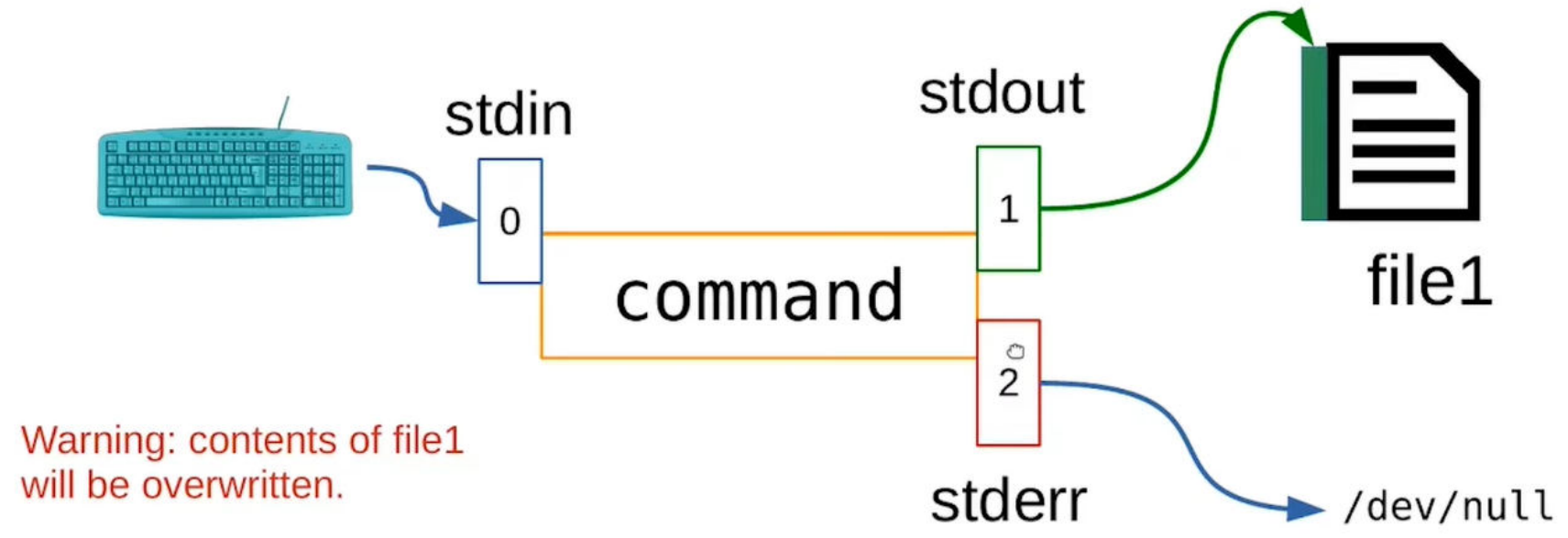
- A sink for output to be discarded
- Use → silent and clean scripts

So, a typical usage looks like ...

`command > file1 2> /dev/null`

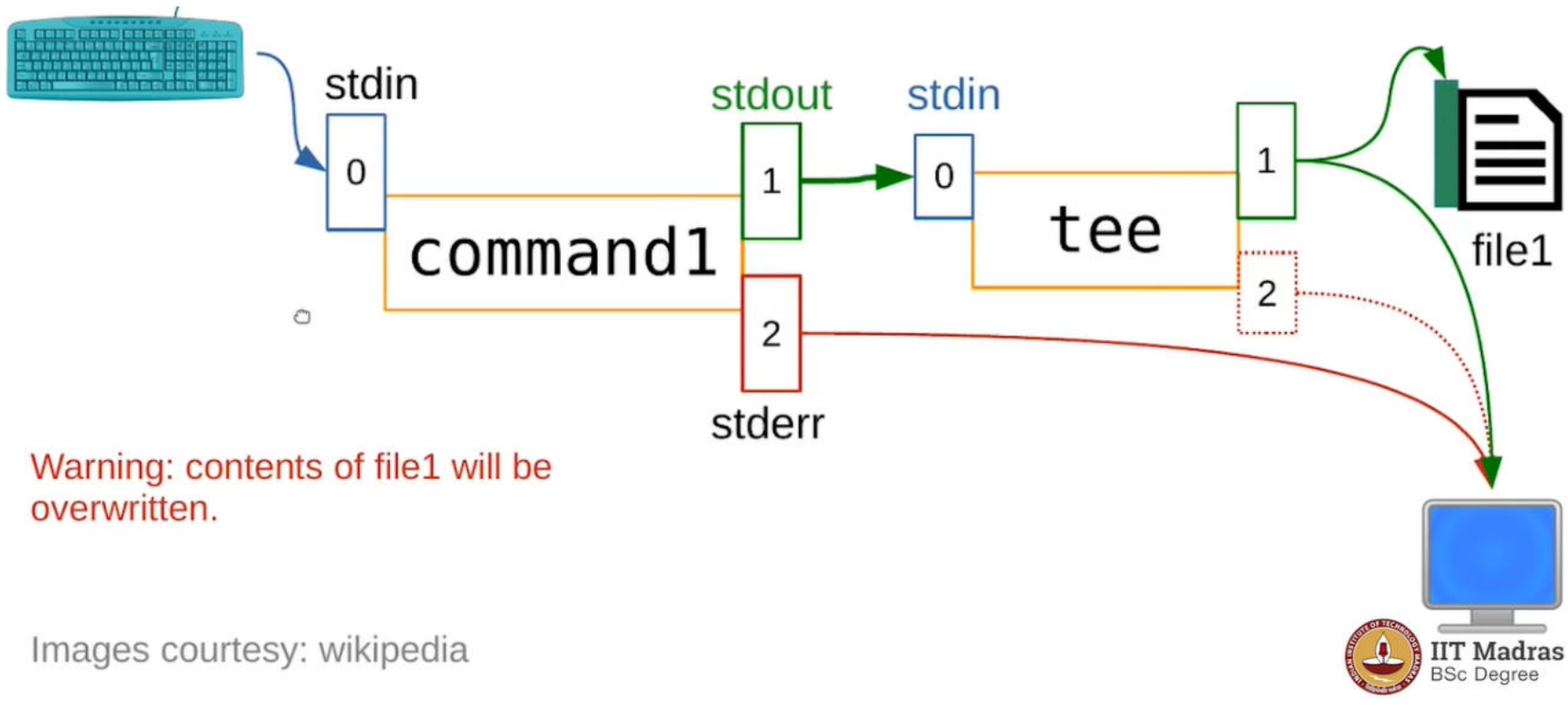
- The output of `command` is written to `file1`

- The `stderr` is written to `/dev/null`, ~~*which gets warped to another dimension*~~



`command1 | tee file1`

- The `tee` command splits the output into 2 streams, one stream is written to the `file1` another, one to the display
  - This command can write to multiple files as well

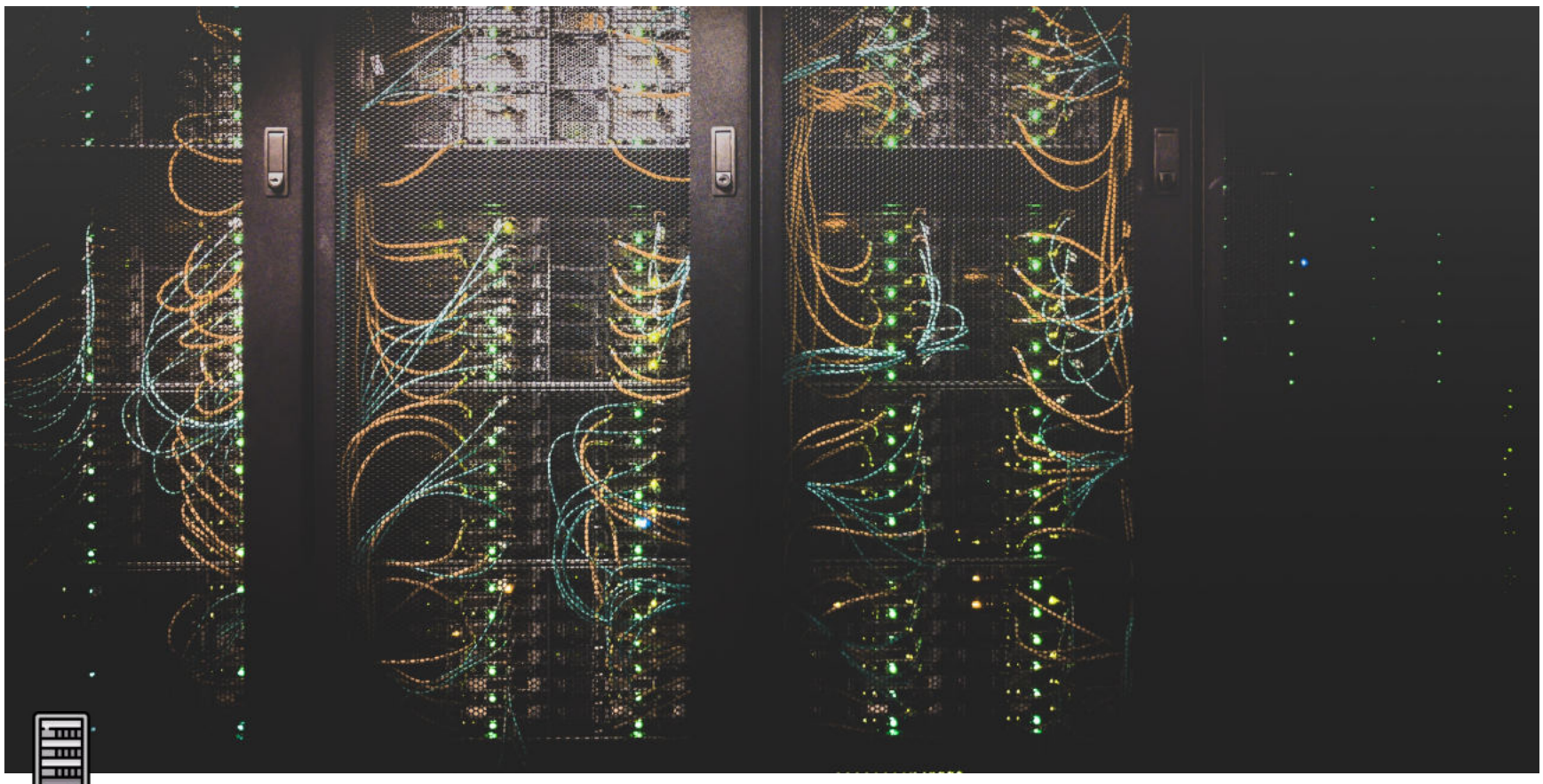


`diff` command

- This command compares files line-by-line

Usage

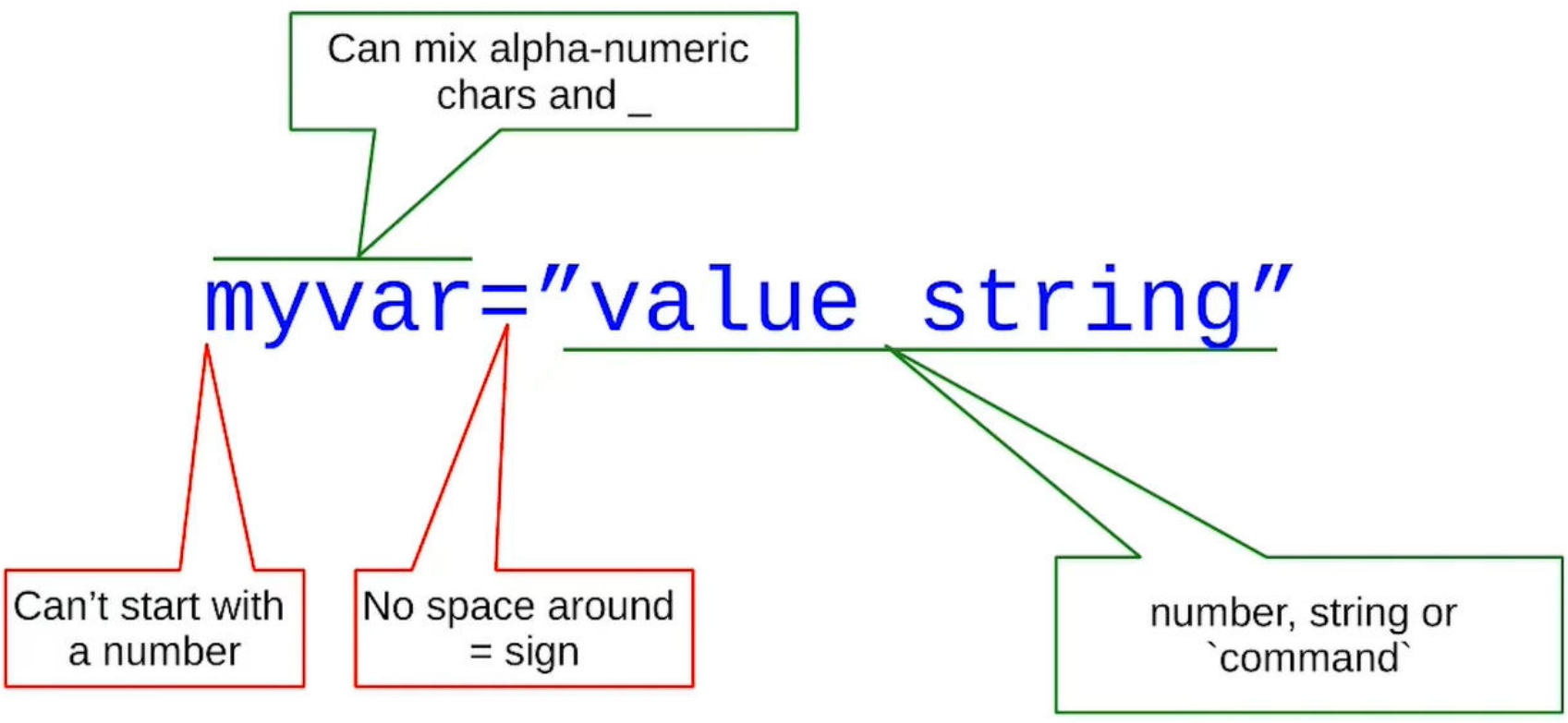
```
diff file1 file2
```



Shell variables

Type	📒 Lecture
Date	@January 9, 2022
Lecture #	3
Lecture URL	https://youtu.be/QX5XElFRpck
Notion URL	https://21f1003586.notion.site/Shell-variables-7fc4b392e99b4d59a13eebefdae88e1e
Week #	3

Creating a variable



Exporting a variable

Exporting means making the value of the variable available to a shell spawned by the current shell (wut)

```
export myvar="value string"
```

OR

```
myvar="value string"
export myvar
```

Using variable values

`echo $myvar`

`echo ${myvar}`

`echo “${myvar}”`

Removing a variable

`unset myvar`

Removing value of a variable

`myvar=`

Test if a variable is set

`[[ -v myvar ]];`

`echo $?`

Return codes:

0 → success (variable `myvar` is set)

1 → failure (variable `myvar` is not set)

Test if a variable is *not* set

`[[ -z ${myvar+x} ]];`

Here, x can be any string

`echo $?`

Return codes:

0 → success (variable myvar is not set)

1 → failure (variable myvar is set)

Substitute default value

If the variable `myvar` is not set, use `“default”` as its default value

`echo ${myvar:-”default”}`

So, if `myvar` *is set, display its value else display “default”*

Reset value if variable is set

If the variable `myvar` is set, then set `“default”` as its value

`echo ${myvar:+”default”}`

So, if `myvar` *is set, change it’s value to “default” and display it else display null*

List of variable names

`echo ${!H*}`

List of names of shell variables that start with `H`

Length of string value

`echo ${#myvar}`

Display length of the string value of the variable `myvar`

If `myvar` *is not set, display 0*

Slice of a string value

`echo ${myvar:5:4}`

Display 4 chars of the string value of the variable `myvar`*, skipping first 5 chars*

Remove matching pattern

`echo ${myvar#pattern}` → *match once*

`echo ${myvar##pattern}` → *match max possible*

Keep matching pattern

`echo ${myvar%pattern}` → *match once*

`echo ${myvar%%pattern}` → *match max possible*

Replace matching pattern

`echo ${myvar/pattern/string}` → *match once and replace with* `string`

`echo ${myvar//pattern/string}` → *match max possible and replace with* `string`

Replace matching pattern by location

`echo ${myvar/#pattern/string}` → *match at beginning and replace with* `string`

`echo ${myvar/%pattern/string}` → *match at the end and replace with* `string`

Changing case

`echo ${myvar,}` → *change the first char to lower case*

`echo ${myvar,,}` → *change all chars to lower case*

`echo ${myvar^}` → *change first char to upper case*

`echo ${myvar^^}` → *change all chars to upper case*

Restricting value types

`declare -i myvar` → *only integers can be assigned*

`declare -l myvar` → *only lower case chars can be assigned*

`declare -u myvar` → *only upper case chars can be assigned*

`declare -r myvar` → *variable is read-only*

You can remove the restrictions by replacing the `-` *sign with a* `+` *sign*

However, `declare +r myvar` will ***NOT*** work

Indexed arrays

`declare -a arr` → *declare* `arr` *as an indexed array*

`$arr[0]="value"` → *set value of element with index 0 in the array*

`echo ${arr[0]}` → *value of the element at index 0 of array*

`echo ${#arr[@]}` → *number of elements in the array*

`echo ${!arr[@]}` → *display all the indices used*

`echo ${arr[@]}` → *display values of all elements of the array*

`unset 'arr[2]'` → *delete element with index 2 in the array*

`arr+=("value")` → *append an element with a value to the end of the array*

Associative arrays

Kind of like Hash maps?

`declare -A hash` → *declare* `hash` *as an associative array*

`$hash["a"]="value"` → *set value of element with index "a" in the array*

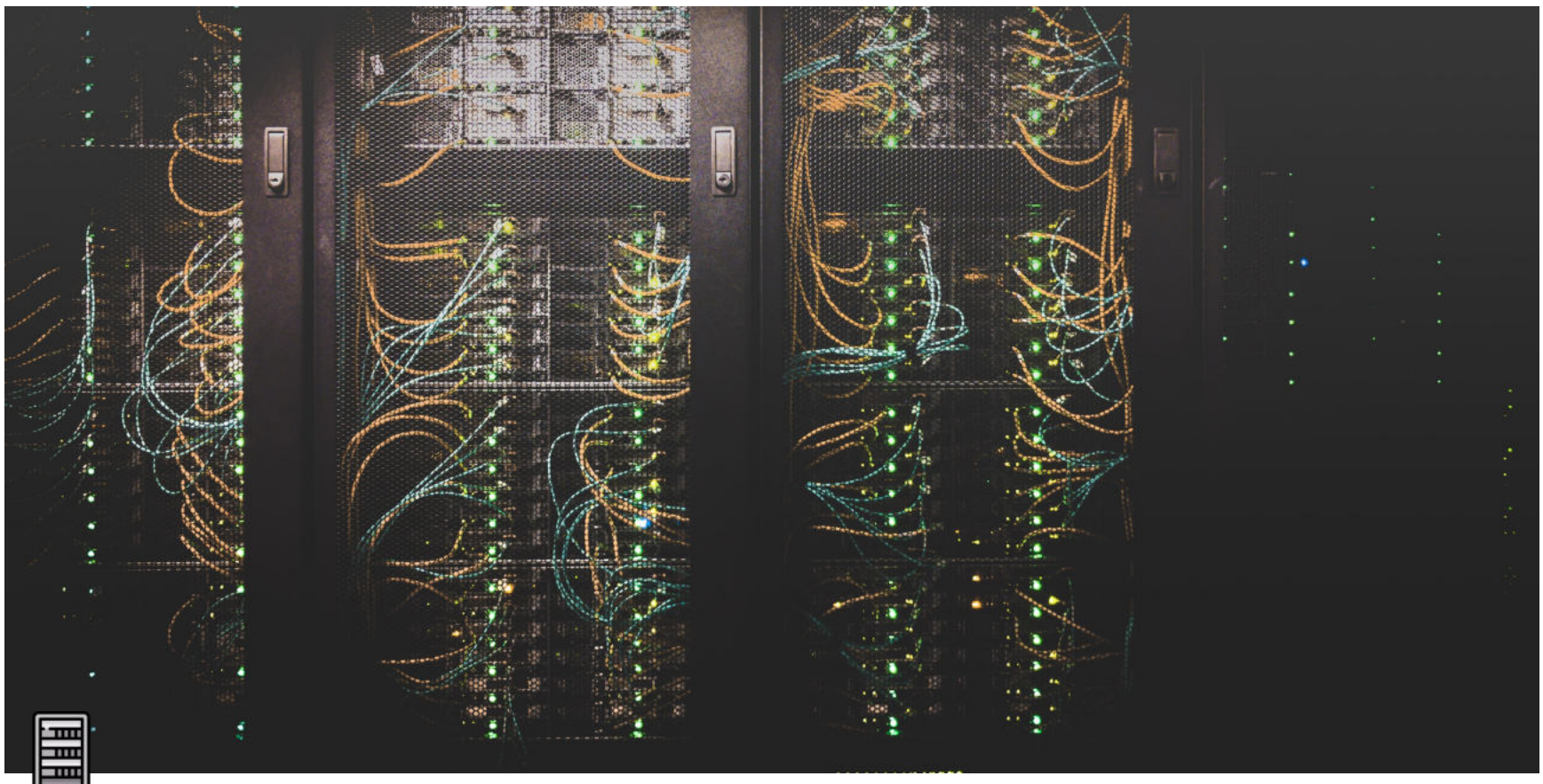
`echo ${hash["a"]}` → *value of element with index (or key?) "a" in the array*

`echo ${#hash[@]}` → *number of elements in the array*

`echo ${!hash[@]}` → *display all indices used*

`echo ${hash[@]}` → *display values of all elements of the array*

`unset 'hash["a"]'` → *delete element with index "a" in the array*



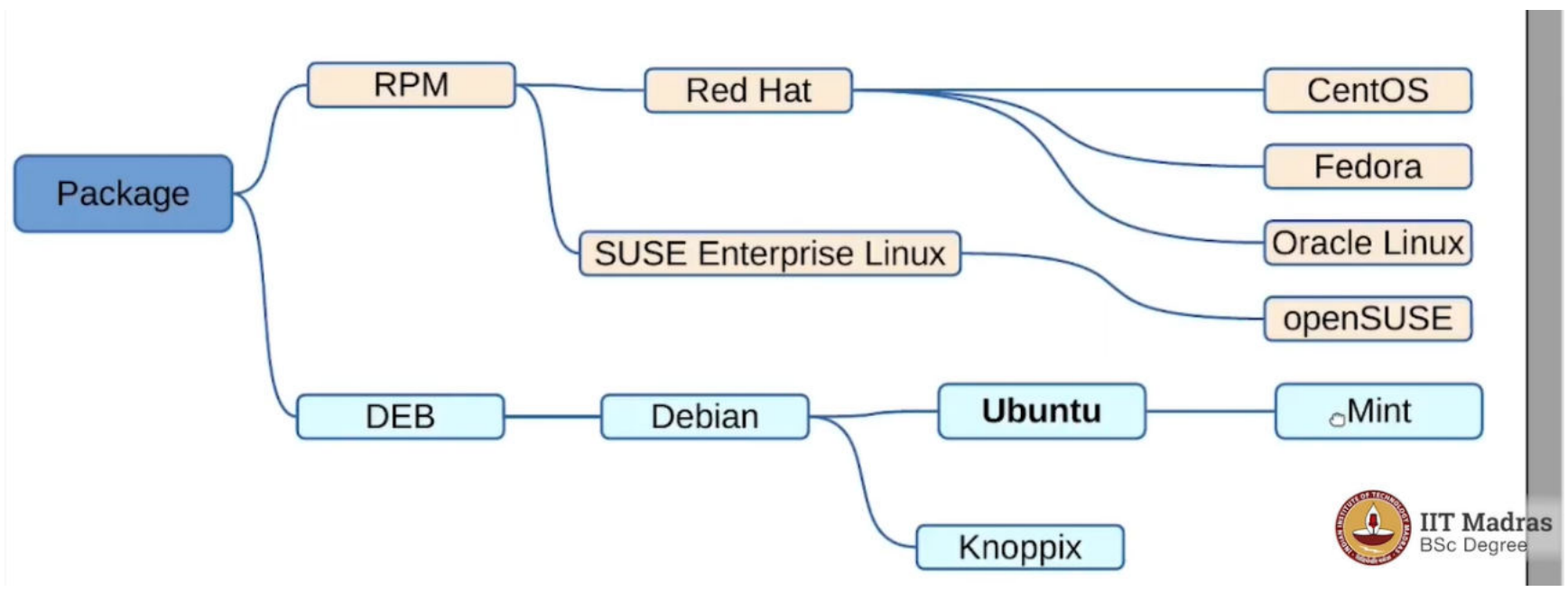
Software Management - pt. 1

Type	📒 Lecture
Date	@January 16, 2022
Lecture #	1
Lecture URL	https://youtu.be/hG-bxCmfArc
Notion URL	https://21f1003586.notion.site/Shell-variables-3c228fce1aef41719f77bc4b8e786ff1
Week #	4

Need for a package manager

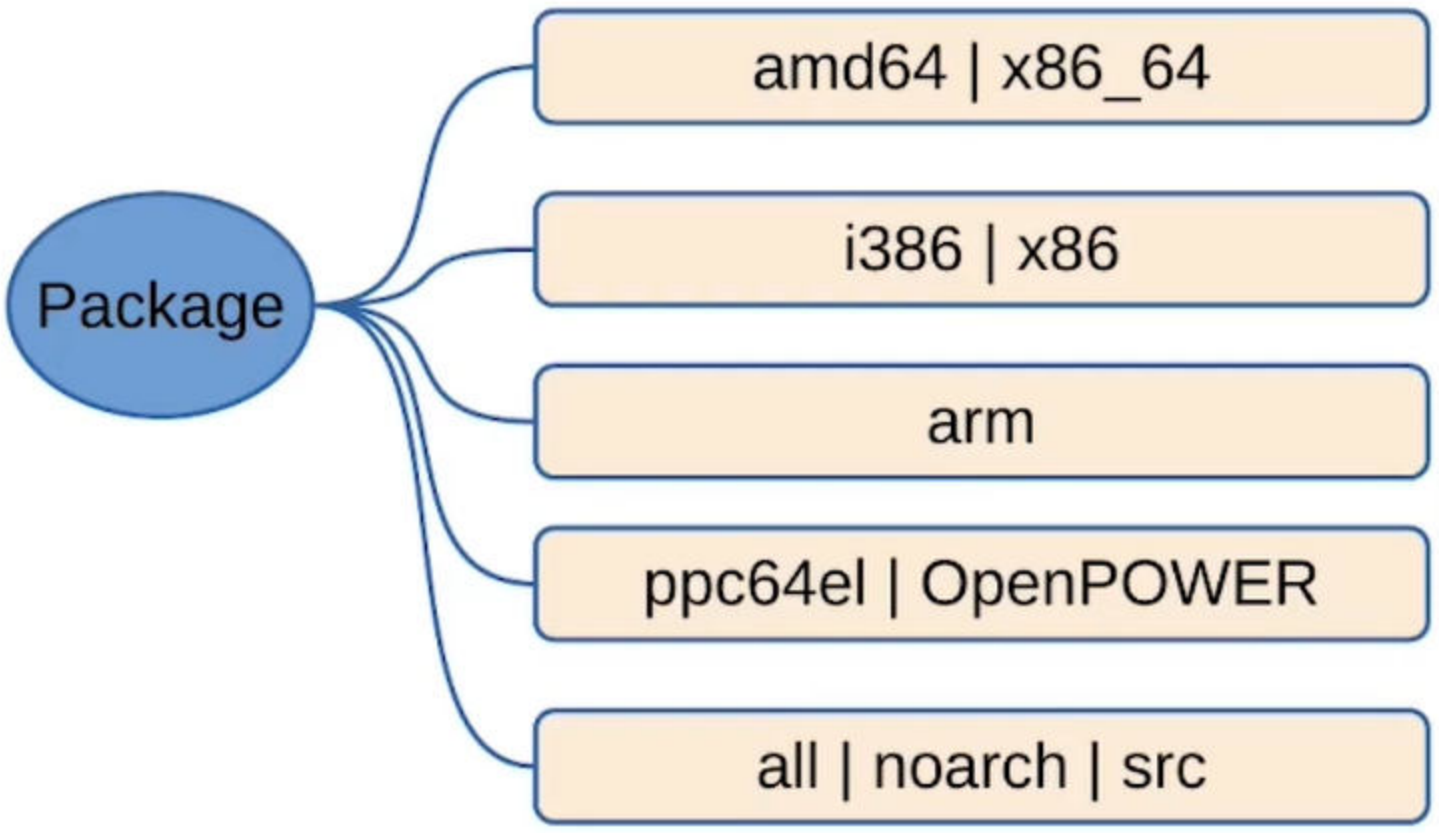
- Tools for installing, updating, removing, managing software
- Install new/updated software across network
- Package — File look up, both ways
- Database of packages on the system including versions
- Dependency checking
- Signature verification tools
- Tools for building packages

Package Types



```
~ lsb_release -a
LSB Version:    n/a
Distributor ID: ManjaroLinux
Description:    Manjaro Linux
Release:        21.2.1
Codename:       Qonos
```

Architecture



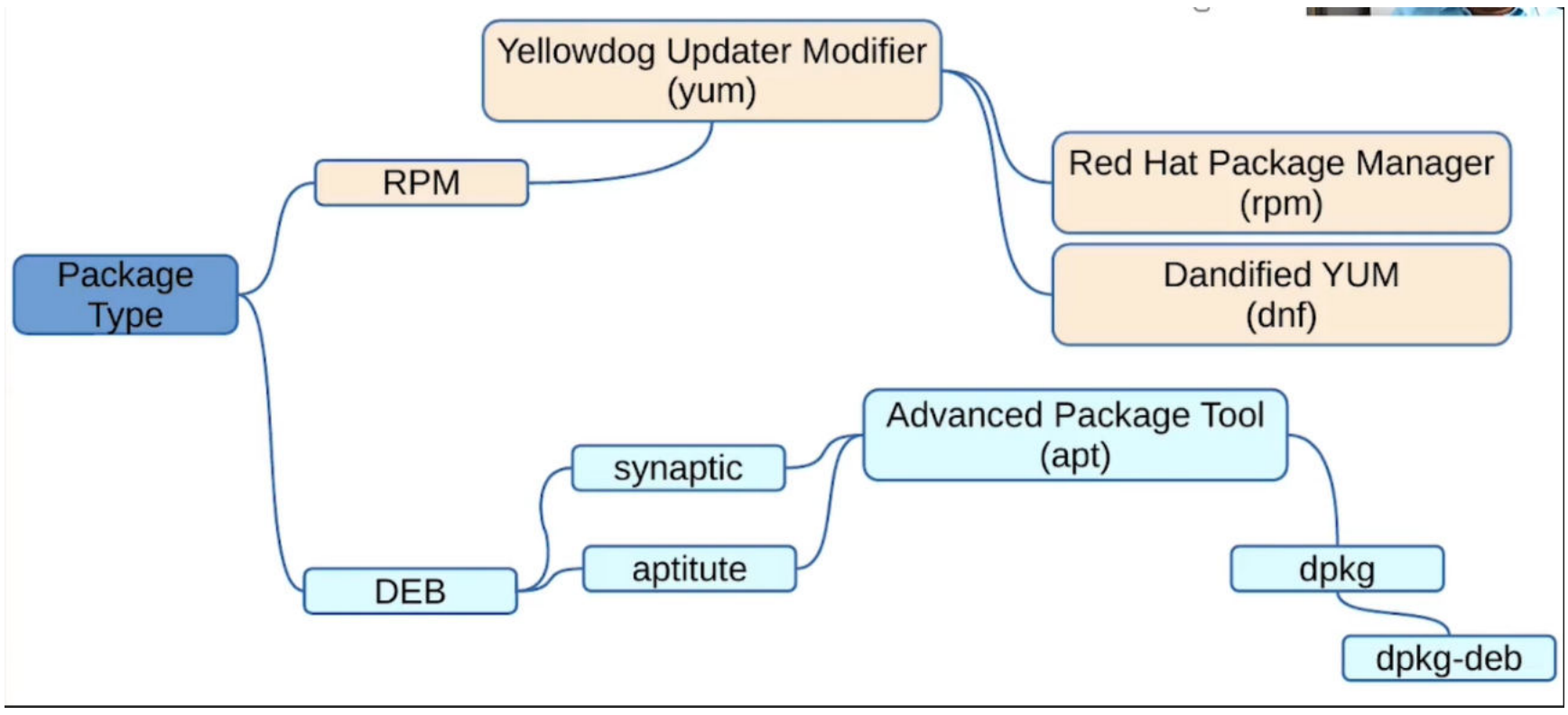
To know your system architecture

~~*Can't spell architecture without arch btw*~~

```
~ uname -a

Linux Zen 5.15.12-1-MANJARO #1 SMP PREEMPT Wed Dec 29 18:08:07 UTC 2021 x86_64 GNU/Linux
```

Tools

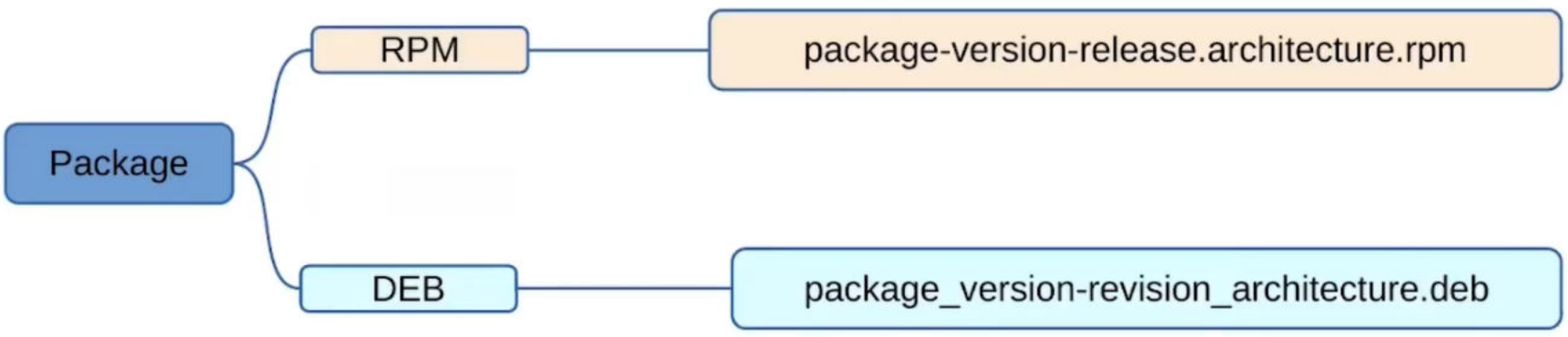


Package Management in Ubuntu using `apt`

Inquiring package db

- Search packages for a keyword
  - `apt-cache search keyword`
- List all packages
  - `apt-cache pkgnames`
  - Try `apt-cache pkgnames <starting-char>` to search to packages with the few starting chars
- Display package records of a package
  - `apt-cache show -a package`

Package names



Package priorities

- **required** → essential to the proper functioning of a system
- **important** → provides functionality that enables the system to run well
- **standard** → included in a standard system installation
- **optional** → can omit if you do not have enough storage
- **extra** → could conflict with packages with higher priority, has specialized requirements, install only if needed

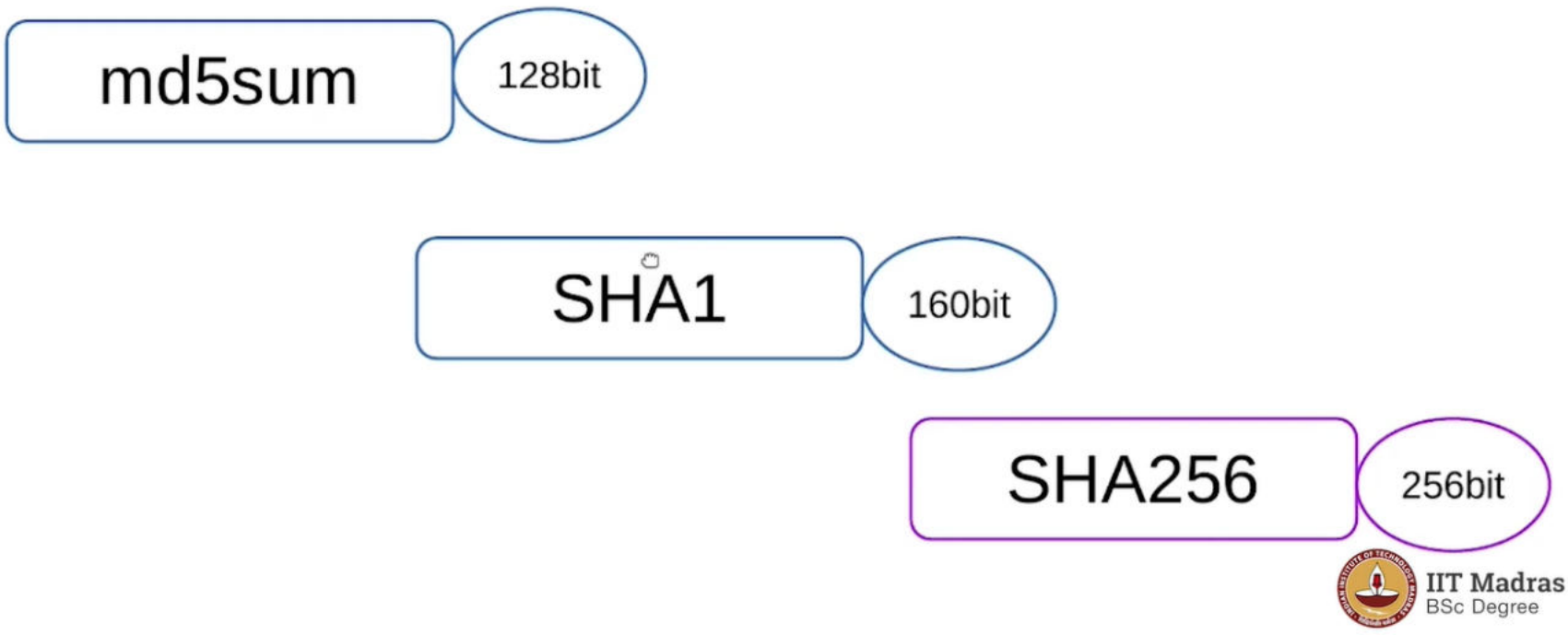
Package sections

https://packages.ubuntu.com/focal

Administration Utilities, Mono/CLI, Communication Programs, Databases, Debug packages, Development, Documentation, Editors, Electronics, Embedded software, Fonts, Games, GNOME, GNU R, GNUstep, Graphics, Haskell, Web Servers, Interpreters, Java, KDE, Kernels, Library development, Libraries, Lisp, Language packs, Mail, Mathematics, Miscellaneous, Network, Newsgroups, OCaml, Perl, PHP, Python, Ruby, Science, Shells, Sound, TeX, Text Processing, Translations, Utilities, Version Control Systems, Video, Web Software, X Window System software, Xfce, Zope/Plone Framework

Checksums

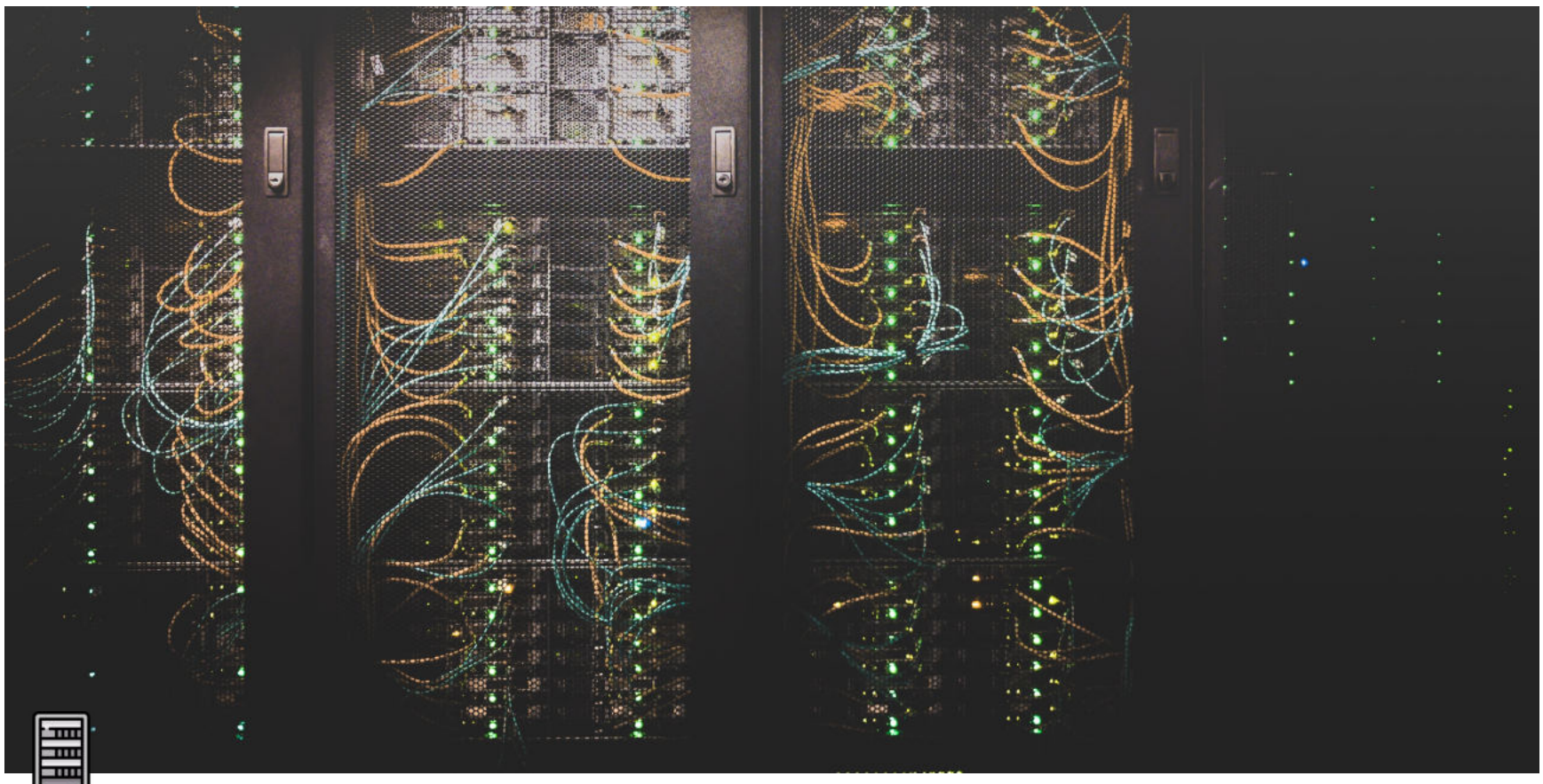
A way to verify that the packages that we are installing are legit



`md5sum <filename>` to get the MD5 checksum of a file, use it to compare it to other files

`sha1sum <filename>` to get the SHA1 checksum of a file, use it to compare it to other files

`sha256sum <filename>` to get the SHA256 checksum of a file, use it to compare it to other files



Software Management - pt. 2

Type	📒 Lecture
Date	@January 16, 2022
Lecture #	2
Lecture URL	https://youtu.be/ctn_s7SDTV0
Notion URL	https://21f1003586.notion.site/Software-Management-b0cd46d7790f479aacf4378fbe9611cf
Week #	4

Only sudoers can install/upgrade/remove packages → `/etc/sudoers`

We can view the `sudoers` file by typing `sudo cat /etc/sudoers`

This will ask for the user password and depending on the `su` status of the user, it will either display the file or show an error like `user is not in the sudoers file. This incident will be reported.`

How can a superuser find out the failed `sudo` attempt by non-su?

Go to `/var/log`

Look for `auth.log` file, these events are reported in this file

Sources for packages

Location: `/etc/apt`

File → `sources.list` → Contains repo URLs for Ubuntu pkgs

Folder → `sources.list.d` → Contains repo URLs for 3rd party pkgs

To get updates

```
sudo apt-get update
```

To upgrade the pkgs (if an upgrade is available)

```
sudo apt-get upgrade
```

You can pass the `-y` flag to not get bugged by the Y/N prompt

To remove older, usually obsolete pkgs

```
sudo apt autoremove
```

To remove a specific pkg

```
sudo apt-get remove <pkg-name>
```

Installing/Updating

- Install a package
  - `apt-get install <pkg-name>`
- Reinstall a package
  - `apt-get reinstall <pkg-name>`

Removing/Cleaning up

- Remove packages that were automatically installed to satisfy a dependency and not needed
  - `apt-get autoremove`
- Clean local repository of retrieved package files
  - `apt-get clean`
- Purge package files from the system
  - `apt-get purge package`

Package management in Ubuntu using `dpkg`

Text info regarding packages found at `/var/lib/dpkg`

Files: `arch, available, status`

Folder: `info`

Using `dpkg`

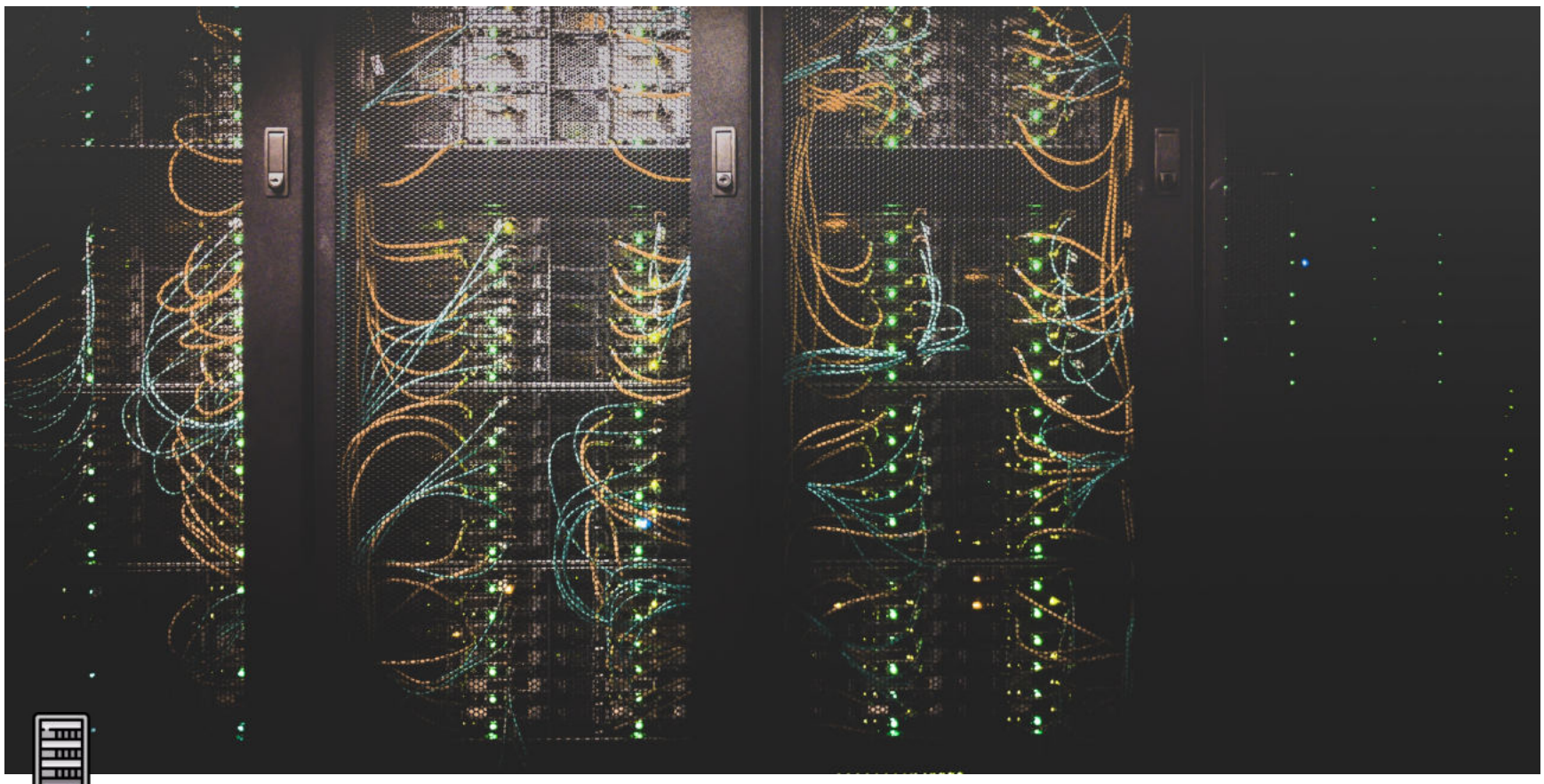
- List all the packages whose names match the pattern
  - `dpkg -l pattern`
- List installed files that came from packages
  - `dpkg -L package`
- Report the status of packages
  - `dpkg -s package`
- Search installed packages for a file
  - `dpkg -S pattern`
- A tool to query the `dpkg` database
  - `dpkg-query`
  - Example usage
    - `dpkg-query -W -f='${Section} ${binary:Package}\n'`

Installing a `deb` package

`dpkg -i filename.deb`

By default, use package management pointing to a reliable repository

Uninstalling packages using `dpkg` is ***NOT*** recommended



Pattern Matching - pt. 1

Type	📒 Lecture
Date	@January 17, 2022
Lecture #	3
Lecture URL	https://youtu.be/1y85iTaqq8Y
Notion URL	https://21f1003586.notion.site/Pattern-Matching-pt-1-8cb1aa5c0e6c42b2a9b517b040006284
Week #	4

Pattern matching

`regex` & `grep`

Regex

- regex is a pattern template to filter text
- BRE: POSIX Basic Regular Expression engine
- ERE: POSIX Extended Regular Expression engine

Why learn regex?

- Languages: Java, Perl, Python, Ruby, ...
  - To process input from the user
  - To perform string operations
- Tools: `grep`, `sed`, `awk`, ...
- Applications: MySQL, PostgreSQL, ...

Usage

- `grep ‘pattern’ filename`
- `command | grep ‘pattern’`
- Default engine: BRE
- Switch to use ERE:
  - `egrep ‘pattern’ filename`
  - `grep -E ‘pattern’ filename`

Special characters (BRE & ERE)

.	Any single character except null or newline
*	Zero or more of the preceding character / expression
[ ]	Any of the enclosed characters; hyphen (-) indicates character range
^	Anchor for beginning of line or negation of enclosed characters
$	Anchor for end of line
\	Escape special characters

Special characters (BRE)

\{n,m\}	Range of occurances of preceding pattern at least n and utmost m times
\(\)	Grouping of regular expressions

Special characters (ERE)

{n,m}	Range of occurances of preceding pattern at least n and utmost m times
( )	Grouping of regular expressions
+	One or more of preceding character / expression
?	Zero or one of preceding character / expression
|	Logical OR over the patterns

Character classes

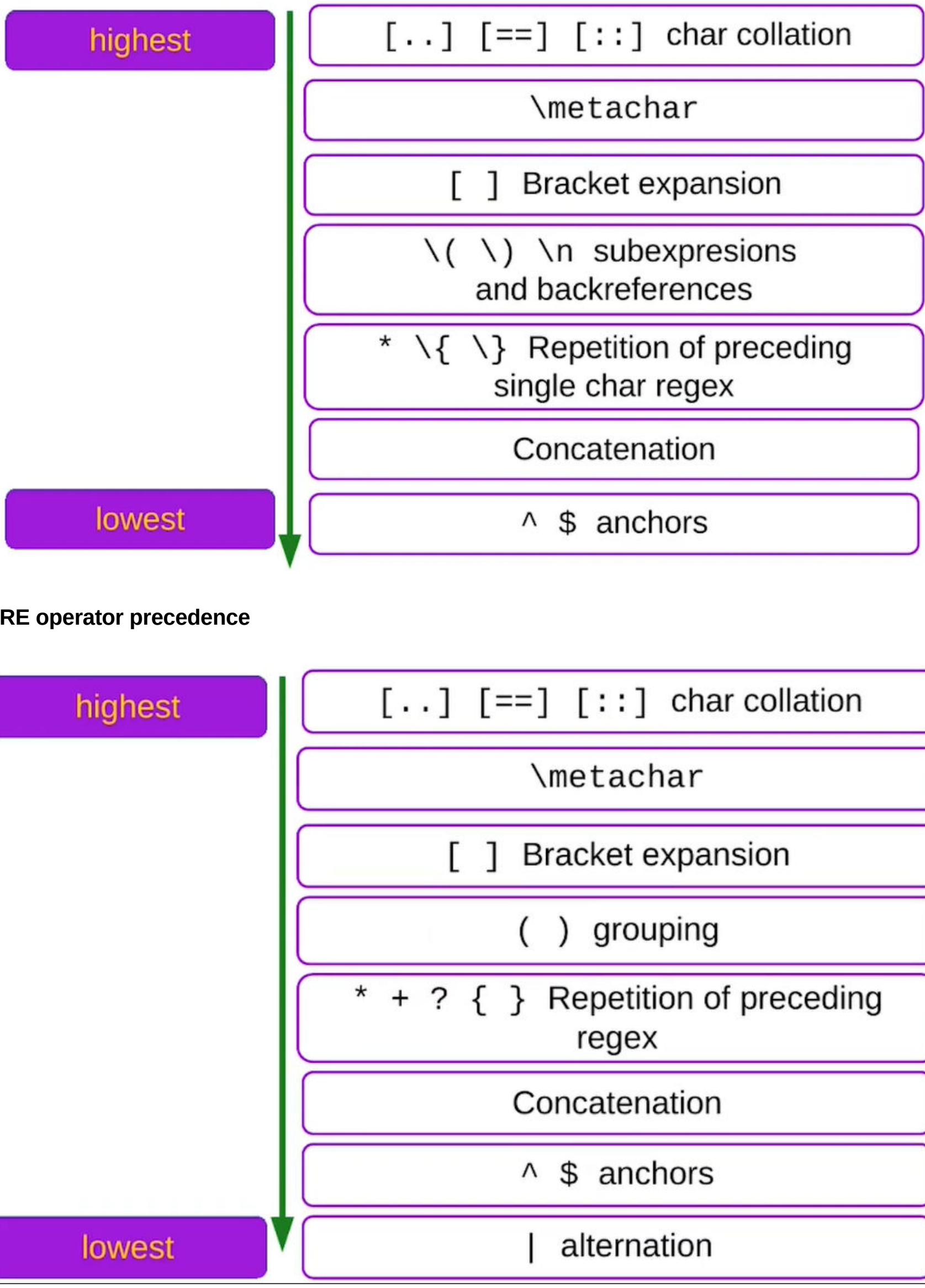
`[[:print:]]`	Printable	`[[:blank:]]`	Space / Tab
`[[:alnum:]]`	Alphanumeric	`[[:space:]]`	Whitespace
`[[:alpha:]]`	Alphabetic	`[[:punct:]]`	Punctuation
`[[:lower:]]`	Lower case	`[[:xdigit:]]`	Hexadecimal
`[[:upper:]]`	Upper case	`[[:graph:]]`	Non-space
`[[:digit:]]`	Decimal digits	`[[:cntrl:]]`	Control characters

To make it slightly easy to match pattern

Backreferences

- `\1` through `\9`
- `\n` matches whatever was matched by the `n`th earlier paranthesized sub-expression
- A line with two occurences of *“hello”* will be matched using: `\(hello\).*\1`

BRE operator precedence



ERE operator precedence

Uses of `grep`

```
~/Documents/week4 cat names.txt
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
~/Documents/week4 grep Raman names.txt
ED22B902 Raman Singh
~/Documents/week4 grep 'Raman' names.txt
ED22B902 Raman Singh
~/Documents/week4 grep 'Anu' names.txt
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
~/Documents/week4 grep 'Sa' names.txt
CS22B904 Charles M. Sagayam
PH22B907 Vel Sankaran
~/Documents/week4 grep 'ai' names.txt
ME22B903 Umair Ahmad
EE22B905 Anu K. Jain
~/Documents/week4 cat names.txt | grep 'ai'
ME22B903 Umair Ahmad
EE22B905 Anu K. Jain
~/Documents/week4
```

Usage of `.` in the `grep` command

`.` is like a wildcard for a single character

```
~/Documents/week4 cat names.txt | grep 'S.n'
ED22B902 Raman Singh
PH22B907 Vel Sankaran
~/Documents/week4 cat names.txt | grep '.am'
MM22B901 Mary Manickam
ED22B902 Raman Singh
CS22B904 Charles M. Sagayam
NA22B906 Anupama Sridhar
```

`$` is used to denote an anchor

Like a pattern at the end of the line

```
~/Documents/week4 cat names.txt | grep '.am$'
MM22B901 Mary Manickam
CS22B904 Charles M. Sagayam
```

Well what if your name contains a `.`

Then escape it using the `\` character

```
~/Documents/week4 cat names.txt | grep '\.'
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
~/Documents/week4
```

If we want the `.` to be necessarily after a character

```
~/Documents/week4 cat names.txt | grep '.\.'
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
~/Documents/week4
```

Match string using anchors at the beginning

ask `grep` to ignore the case by passing in the `-i` flag

```
~/Documents/week4 cat names.txt
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
~/Documents/week4 cat names.txt | grep '^M'
MM22B901 Mary Manickam
ME22B903 Umair Ahmad
~/Documents/week4 cat names.txt | grep '^E'
ED22B902 Raman Singh
EE22B905 Anu K. Jain
~/Documents/week4 cat names.txt | grep '^e'
~/Documents/week4 cat names.txt | grep -i '^e'
ED22B902 Raman Singh
EE22B905 Anu K. Jain
~/Documents/week4
```

Match a pattern at the end of the line, a word boundary one might say

`\b` looks for a word boundary, so that pattern could also occur at the end of a word in the middle of a line

`$` looks for line boundary only, so the pattern occurs at the end of the line

```
~/Documents/week4 cat names.txt
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
~/Documents/week4 cat names.txt | grep 'am\b'
MM22B901 Mary Manickam
CS22B904 Charles M. Sagayam
~/Documents/week4 cat names.txt | grep 'am$'
MM22B901 Mary Manickam
CS22B904 Charles M. Sagayam
~/Documents/week4
```

Usage of square brackets `[]`

Here, the first character in the pattern is followed by either of the 2 characters given in `[]`

In the `grep 'S.*[mn]'` command, it matches from the start of the line

We add `\b` to mark a word boundary just to match it within a word

Had to switch to `bash` to show the formatting

```
[kashif@Zen week4]$ cat names.txt
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
[kashif@Zen week4]$ cat names.txt | grep '[ME]E'
ME22B903 Umair Ahmad
EE22B905 Anu K. Jain
[kashif@Zen week4]$ cat names.txt | grep 'E[ED]'
ED22B902 Raman Singh
EE22B905 Anu K. Jain
[kashif@Zen week4]$ cat names.txt | grep 'M[EM]'
MM22B901 Mary Manickam
ME22B903 Umair Ahmad
[kashif@Zen week4]$ cat names.txt | grep 'S.*[mn]'
ED22B902 Raman Singh
CS22B904 Charles M. Sagayam
PH22B907 Vel Sankaran
[kashif@Zen week4]$ cat names.txt | grep '\bS.*[mn]'
ED22B902 Raman Singh
CS22B904 Charles M. Sagayam
PH22B907 Vel Sankaran
```

```
[kashif@Zen week4]$ cat names.txt
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
[kashif@Zen week4]$ cat names.txt | grep '[aeiou]'
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
[kashif@Zen week4]$ cat names.txt | grep '[aeiou][aeiou]'
ME22B903 Umair Ahmad
EE22B905 Anu K. Jain
[kashif@Zen week4]$ cat names.txt | grep 'B90[1-4]'
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
[kashif@Zen week4]$ cat names.txt | grep 'B90[5-7]'
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
[kashif@Zen week4]$ cat names.txt | grep 'B90[1-9]'
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
[kashif@Zen week4]$
```

The last command in the following screenshot does negation

```
[kashif@Zen week4]$ cat names.txt
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
[kashif@Zen week4]$ cat names.txt | grep '[M-Z][aeiou]'
MM22B901 Mary Manickam
ED22B902 Raman Singh
CS22B904 Charles M. Sagayam
PH22B907 Vel Sankaran
[kashif@Zen week4]$ cat names.txt | grep 'B90[^5-7]'
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
[kashif@Zen week4]$
```

Number of times a character should occur

In the curly braces, we provide the # of times the preceding character should be matched

We can pass one number or a multiple numbers separated by comma for their matching

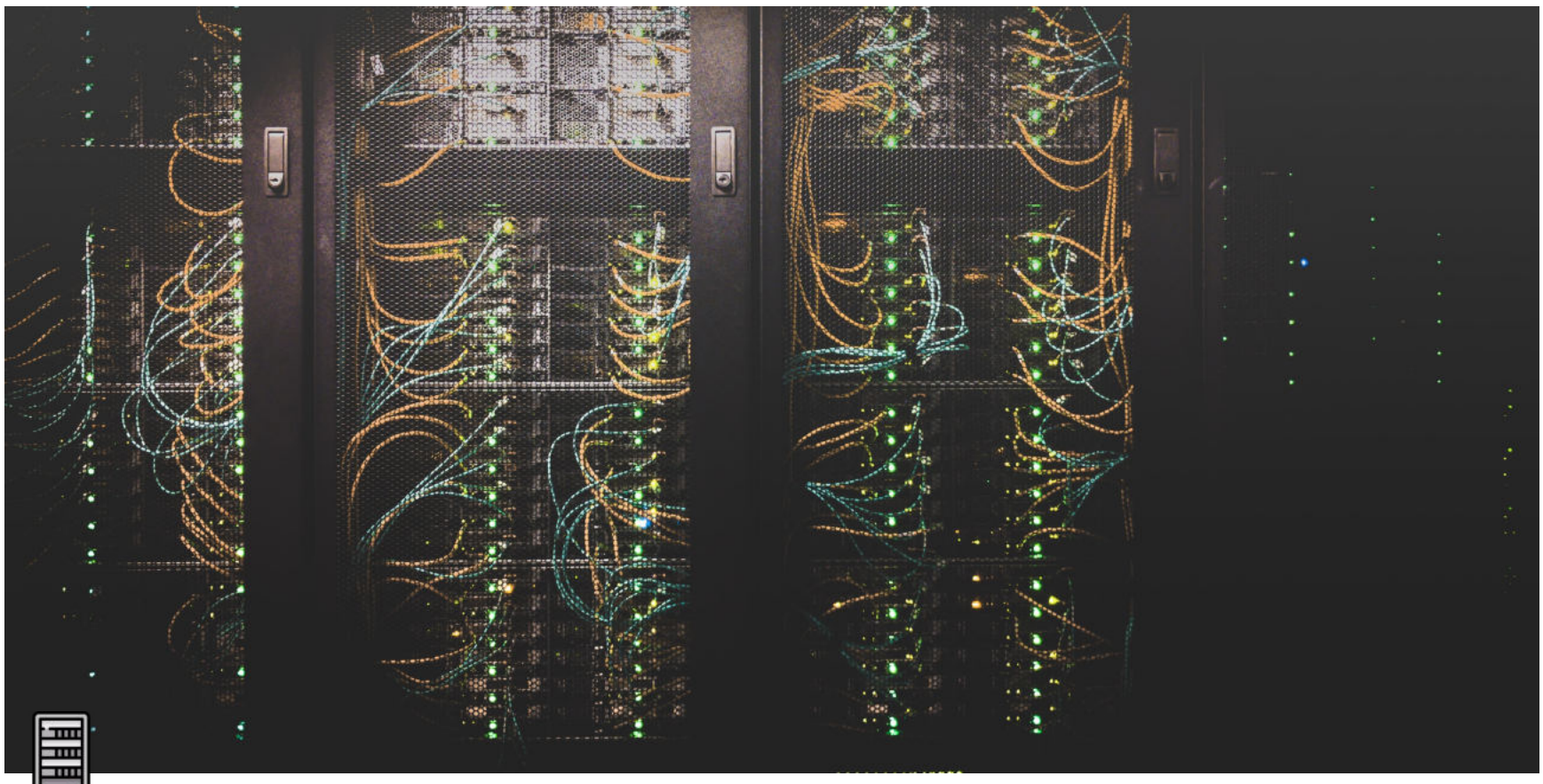
```
[kashif@Zen week4]$ cat names.txt
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
[kashif@Zen week4]$ cat names.txt | grep 'M\{2\}'
MM22B901 Mary Manickam
[kashif@Zen week4]$ cat names.txt | grep 'M\{1,2\}'
MM22B901 Mary Manickam
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
[kashif@Zen week4]$
```

```
[kashif@Zen week4]$ cat names.txt
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
[kashif@Zen week4]$ cat names.txt | grep '\(ma\)'
ED22B902 Raman Singh
ME22B903 Umair Ahmad
NA22B906 Anupama Sridhar
[kashif@Zen week4]$ cat names.txt | grep '\(ma\).*\1'
ME22B903 Umair Ahmad
[kashif@Zen week4]$ cat names.txt | grep '\(.a\).*\1'
MM22B901 Mary Manickam
ME22B903 Umair Ahmad
[kashif@Zen week4]$ cat names.txt | grep '\(a.\).*\1'
PH22B907 Vel Sankaran
[kashif@Zen week4]$ cat names.txt | grep '\(a.\)\{3\}'
CS22B904 Charles M. Sagayam
[kashif@Zen week4]$ cat names.txt | grep '\(a.\)\{2\}'
ED22B902 Raman Singh
CS22B904 Charles M. Sagayam
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
[kashif@Zen week4]$ cat names.txt | grep '\(a.\)\{2,3\}'
ED22B902 Raman Singh
CS22B904 Charles M. Sagayam
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
[kashif@Zen week4]$
```

```
[kashif@Zen week4]$ cat names.txt
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
[kashif@Zen week4]$ cat names.txt | egrep 'M+'
MM22B901 Mary Manickam
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
[kashif@Zen week4]$ cat names.txt | egrep '^M+'
MM22B901 Mary Manickam
ME22B903 Umair Ahmad
[kashif@Zen week4]$ cat names.txt | egrep '^M*'
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
[kashif@Zen week4]$ cat names.txt | egrep 'M*a'
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
[kashif@Zen week4]$ cat names.txt | egrep 'M.*a'
MM22B901 Mary Manickam
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
[kashif@Zen week4]$
```

```
[kashif@Zen week4]$ cat names.txt
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
[kashif@Zen week4]$ cat names.txt | egrep '(ma)+'
ED22B902 Raman Singh
ME22B903 Umair Ahmad
NA22B906 Anupama Sridhar
[kashif@Zen week4]$ cat names.txt | egrep '(ma)*'
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
[kashif@Zen week4]$
```

```
[kashif@Zen week4]$ cat names.txt
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
[kashif@Zen week4]$ cat names.txt | egrep '(ED|ME)'
ED22B902 Raman Singh
ME22B903 Umair Ahmad
[kashif@Zen week4]$ cat names.txt | egrep '(Anu|Raman)'
ED22B902 Raman Singh
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
[kashif@Zen week4]$ cat names.txt | egrep '(am|an)'
MM22B901 Mary Manickam
ED22B902 Raman Singh
CS22B904 Charles M. Sagayam
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
[kashif@Zen week4]$ cat names.txt | egrep '(am|an)$'
MM22B901 Mary Manickam
CS22B904 Charles M. Sagayam
PH22B907 Vel Sankaran
[kashif@Zen week4]$
```



Pattern Matching - pt. 2

Type	📒 Lecture
Date	@January 17, 2022
Lecture #	4
Lecture URL	https://youtu.be/XQUJPRc-7zA
Notion URL	https://21f1003586.notion.site/Pattern-Matching-pt-2-18da9c08be534558ad86e28d07cba0bd
Week #	4

Match package names that are 4 characters long

```
dpkg-query -W -f='${Section} ${binary:Package}\n' | egrep ' .{4}$'
```

Match package names that are 3 characters long and start with the letter `g`

```
dpkg-query -W -f='${Section} ${binary:Package}\n' | egrep ' g.{3}$'
```

Match package names that are between 1 to 5 characters long and start with the letter `g`

```
dpkg-query -W -f='${Section} ${binary:Package}\n' | egrep ' g.{1,5}$'
```

Match package names that are from the `math` category

```
dpkg-query -W -f='${Section} ${binary:Package}\n' | egrep '^math'
```

make sure to use the `^` (hat) character in the front of the regex pattern to match the `math` category, otherwise it will match package category and the names

Match package names that from KDE

```
dpkg-query -W -f='${Section} ${binary:Package}\n' | egrep ' kd.*$'
```

To skip empty lines from a file

```
cat filename.txt | egrep -v '^$'
```

- Pick any 12 digit or more number from a text file
  - `egrep '[[:digit:]]{12}' filename.txt`
- Pick any 6 digit or more number from a text file
  - `egrep '[[:digit:]]{6}' filename.txt`

But, there is one problem, if there is any number that is more than 12 digits or more than 6 digits respectively, it will pick that up too

- Pick an exactly 6 digit number from a text file
  - Add a word boundary `\b`

    - `egrep '\b[[:digit:]]{6}\b' filename.txt`
- Pick a roll number (of the type MM22B001) from a text file
    - `egrep '\b[[:alpha:]]{2}[[:digit:]]{2}[[:alpha:]][[:digit:]]{3}\b' filename.txt`
- Pick a URL from a text file (like github.com or https://www.iitm.ac.in)
    - `egrep '\b[[:alnum:]]+\.[[:alnum:]]+\b' filename.txt`

`cut`

A command used to cut lines from files

does horizontal trimming

A sample file `fields.txt`

```
1234;hello world,line-1
234567;welcome cmdline,line-2
3456;parse text,line-3
```

- Cut first 4 characters from the beginning of the lines
    - `cut -c 1-4 fields.txt`
- Cut the next 4 characters from the previous
    - `cut -c 5-8 fields.txt`
- We can skip the beginning or the ending of the substring parameter, *it works like python*
    - Cut 4 chars from the beginning
        - `cut -c -4 fields.txt`
    - Cut from 8th char to the end
        - `cut -c 8- fields.txt`
- Use space as the delimiter and print the first field
    - `cat fields.txt | cut -d " " -f 1`
- Similarly, print the second field
    - `cat fields.txt | cut -d " " -f 2`
- If we want both fields
    - `cat fields.txt | cut -d " " -f 1-2`
- Delimit at a semi-colon `;` and get the first field
    - `cat fields.txt | cut -d ";" -f 1`
- Similarly, get the 2nd field
    - `cat fields.txt | cut -d ";" -f 2`
- We can pipe multiple commands
    - To get the part of the line between `;` and `,`
        - `cat fields.txt | cut -d ";" -f 2 | cut -d "," -f 1`
    - To do the same thing using `grep` (*similar thing, not exactly the same*)
        - `cat fields.txt | egrep ';.*,'`
- To get the part `welcome cmdline` from the file `fields.txt`
    - `cat fields.txt | cut -d ";" -f 2 | cut -d "," -f 1 | head -n 2 | tail -n 1`