Open a **terminal** Try press **ctrl + alt + t**

By default many terminals will popup with so called 'bash'. To see which shell you are using under the hood, type:

echo $SHELL

If the response is **/bin/bash**, you are using 'bash'. Or in short your shell is whatever you have after **/bin/**.

I prefer to use **zsh** with **oh.my.zsh**. Rest of the tutorial will focus on **zsh** usage.

Try typing: **id**

Then try: **id -h**

Play around with parameters and observe what the function returns.

Similarly try: **uname** and **uname -A**

On a similar token, try:

who

whoami

To get a quick summary of what's up try: **w**

Ok but, where are the files?

If you still remember MS DOS, MS DOS equivalent is **dir** and works here as well, try and observe the difference with **l**is**t** of the folder content: **ls**

Awesome there are colors, but why are there colors? Well, not necessarily why, but it tells us that people must be spending a lot of time using the terminal :)

How do I change directory then? Choose one from the list and go into it **cd directory_u_choose**

List the content of your folder via **ls**

Now go back to where you were **cd ..**

What does '**..**' do?

This time try **ls -a**

Also try **ls -A**

What is different between **ls** and **ls -a** and **ls -A**?

Next try; **ls -l**

and: `ls -al`

What is the dash '**-**' for ?

Well what are the options that are available?

Let's get a **man**ual for the command **ls**: `man ls`

User arrow keys to browse and type '**q**' to quit.

If you just want a brief explanation of what a command is intended for try: `whatis command_name`

For example: `whatis ls`

Boy, almost the whole alphabet is here for option, and lower and upper cases mean different things!

IMPLICIT LESSON: Dealing with Linux will make you read a lot at first, and potentially then... :)

But is will eventually grow on you :) or you will hate it :(- You will live to decide...

Now let's create a folder, or in other words make a directory, named say, test1: `mkdir test1`

Make another test directory, named say, test2: `mkdir test2`

But change into the first: `cd test1`

Now change into `test2`: `cd ../test2`

Works like a charm, but assume that after changing `test2` you did a lot of work and need to go back to `test1`. Also assume that `test1` and `test2` are on very different branches. Good thing is you can always go back to the last folder you were in (namely the old folder, or using the name of the environment variable that holds it: OLDPWD), try: `cd -`

In `test1` make two more folders in here: `mkdir test11; mkdir test12`

Type `cd te` and press **tab**, what has just happened?

Press **tab** two more times to get alternatives and change in to test12: `cd test12`

I am lost, where am I? In other words, what is the **p**resent **w**orking **d**irectory? `pwd`

Clear screen when it gets messy with: `clear`

Type in some meaningless characters and press **enter** so that the screen gets messy, and rather than `clear` this time try `ctrl-l` (i.e. *control key* and *lower case L*). Observe that this shortcut does the same thing.

Hmm there are shortcuts… cool…

Couple of more use full shortcuts:

`ctrl-u`: Erase all the way to the beginning

`ctrl-k`: Erase all the way to the end

`ctrl-w`: Erase backwards word by word

`ctrl-y`: Yank what you have erased (just like paste, but not paste, try out and understand the difference)

`ctrl-a`: Go to the beginning of line

`ctrl-e`: Go to the end of line

So long shortcuts, let's continue… Where were we?

To remember where I am on an empty screen, one more time **pwd**. Oops, now **pwd** command shows up, still not a clear reminder! How about two commands on a single line: **clear; pwd**

Semi column is with us in the terminal… Apparently you can separate multiple commands using it. Would not it be nice if there was a single command that does "**clear; pwd**" altogether. Or would not it be nice if I could write my custom command such as "**clear; pwd**" and give it some name I like and use it to mean "**clear; pwd**" anytime I want? Then wait until aliases ☺

Haven't you already started missing the ease of mouse and navigating via clicking… Please say **Noooo**! Not because I would like, but because if you work on your typing skills, many information can be reached quicker, and many tasks can be finished faster via the terminal.

Given the present working directory, what is the difference between **pwd** and the bash prompt?

Open a fresh terminal to see the initial bash prompt: press **ctrl + alt + t**

Ok, when it opens fresh, this is your **HOME**

Wow, why does all **pwd** results start with '**/**' ?

Because it is the **root** in your file system. Try: **cd /**

Take a list: **ls**

Now close one of the terminals via: **exit**

How does the system know, say, where my home is? Really, where is my home? Am I lost? heelp…

Try: **printenv**

to print the environment variables. (also try **env**)

Oooh, my eyes and brain hurt… too many lines… how do I know if there is a **HOME** in here?

Now try: **printenv | grep HOME**

So, HOME is like a system variable! Browse through the environment variables…

You can also use **egrep** which is similar to **grep**.

Well, what if (for some odd reason) you would like to save the content of these variables to a file and read them later. But first go make sure that you are in folder **test12**. Go to this folder on your own now.

printenv > test_env_vars.txt

Find if the word USER is in this file (read more on environment variables):

cat test_env_vars.txt | grep USER

Here **cat** simply prints the content of the file and **grep** grabs the lines that contains USER

To see the full content of the file type:

cat test_env_vars.txt

In case you need to learn the number of a line along with the content try:

cat -n test_env_vars.txt

Even better you can combine **cat -n** with **grep** to find the line number of what you are looking for:

cat -n test_env_vars.txt | grep *keyword*

For some odd reason, if you want to see the see the lines in reverse order (i.e. listed in a bottom up fashion) try:

tac test_env_vars.txt

tac might not be available in MacOS and Windows terminals

Let's create a simple file in this folder. Type:

echo "Are we there yet?" > testere

Check the content of the file with cat: **cat testere**

Observe that extension did not matter! Now add another line to the file

echo "No son" > testere

Check the content of the file with cat: **cat testere**

What happened to the previous line?

Now try the following:

echo "Are we there yet?" > testere

echo "Will be there soon son…" >> testere

Check the content of the file with cat: **cat testere**

How about a simple editor, that not only shows the content, but lets me edit files on the terminal, try: **pico** or **nano**.

These are not the best editors, but they get the job done.

Now try more and less:

more testere

less testere

In **more**, press space for page down.

In in **less** use up / down arrows, and press '**q**' to quit.

Now try the same with more or less for the text file:

more test_env_vars.txt

less test_env_vars.txt

What if you are not interested in the whole file

head test_env_vars.txt

tail test_env_vars.txt

If default number of lines is not good, try –n. Following example will let you peek into the first and last 3 lines in **test_env_vars.txt** respectively.

head -3 test_env_vars.txt

tail -3 test_env_vars.txt

In case you need the number of lines, words or characters try **w**ord **c**ount command: **cat test_env_vars.txt | wc**

It should be expected that you can only get number of lines, words or characters. Read the help of **wc** for more info.

If this is an environment variable, given I know the name of the variable, can I see its content? Try:

printenv $USER

Here **$** does get the value of a variable. Now try something new but with similar result:

echo $USER

echo command is for printing arguments to standard output.

Try a simple, and pretty useless echo as:

echo hello dude

Now create a custom variable: **x=5** (No spaces!)

echo $x

Yet using terminal as a calculator is not your best option.

Talking about which, is there a calculator in the terminal? Simply type and try:

calculator

and be amazed … linux smart mouths you … install galculator? :

```
No command 'calculator' found, did you mean:  Command 'galculator' from
package 'galculator' (universe) calculator: command not found
```

What to do when I need something that is not installed… Hmmm, there is no recommendation or anything…

So let's be patient but keep this in mind.

Will come back to galculator later but now type: **sl**

Disappointed padawan? Not be! Linux there to help you, just the hint to read :]

And try this educated guess:

sudo apt install sl

It is worth to note that, previously it was **apt-get** but now **apt** works just well enough.

To run it, simply type: **sl**

Enjoy the ride…

Here **apt install** manages the installation. **apt ***** manages other things as well. You will see, need and use **apt ***** very often.

sudo right before **apt install** give you **s**uper **u**ser privileges so that you can do super duper things.

Beware when you are Super: *power corrupts, absolute power absolutely corrupts!* →Messing up takes almost no time, fixing it sometimes takes days, if not worse.

Could I have done the same trick with `'galculator'` when I needed the calculator? Try as suggested:

sudo apt install galculator

and simply type **galcu** and tab to complete, to run it.

Wow, a GUI for the first time. Now check what happened to the terminal! As long as calculator is open, the *terminal is busy* with it and you cannot use it. Either open a new one, or simply kill this calculator app via the GUI or press **ctrl + c**.

For fun try the following, this time two packages installed at the same time:

sudo apt install fortune cowsay

Try **fortune** command couple of times: **fortune**

You got the idea. Now do the same for cowsay: **cowsay**

Crap! How do I get out of this? Try **ctrl + c**

Remember, **ctrl + c** will get you out of places where you do not want to stay!

Try: **man cowsay**

When you are satisfied with knowledge try: **cowsay "I got milk"**

Now combine the output of fortune with cowthink via pipe (i.e. '**|**'): **fortune | cowthink**

Just for the fun of it and to stick with the theme try: **cowthink -f tux "linux rocks"**

To get the full listing of alternatives, finally try: **cowthink -l**

Experiment with some of the characters…

If you are into more meaningless terminal fun, try out: **banner** and **figlet**. Now you know what to do if they are not installed.

Once you are bored with these terminal quirks or you think you do not need a particular package you can always:

sudo apt remove package_name_as_installed

When you keep installing programs and creating files, at on point you will wonder how much disk space is free, in that case: **df**

To see the result in mega bytes try: **df -m**

In case you need to compress a file, **gzip** is a command line compression tool. **gzip filename** will compress, **gzip -d filename** will decompose (i.e. *unzip*).

The screen should be messy by now, finally clear the screen with: **clear**

Other similar purpose **clear** commands in different IDEs are **clc**, **clr** or **cls**, all are shorter than **clear**:)

Try **clc**, **clr** or **cls** and observe that it fails!

How can I define them to work like **clear**?

Welcome the **alias** almighty!

Create an alias as clr by typing: **alias clr='clear'**

Do stuff to make the screen messy…

Now type: **clr**

Just for keeping our prior promise, define: **alias cls='clear; pwd'**

Try it. Now that you have more than one alias, it already begun to get complicated.

What is you have tons of aliases? Try typing: **alias**

Awesome ain't it? Now type: **exit**

Open a new terminal and type: **clr**

Hmm, to make things sticky, terminals should remember them. Everytime a terminal is opened, contents of **.bashrc** file will be executed. Hence, we can edit **.bashrc** file, but we do NOT want to edit it too much since it looks complicated and if we mess it, it might mess up how the terminal behaves.

At this point, how about a better editor, such as Sublime Text (2 or 3)

For Sublime-Text-2:

```
sudo add-apt-repository ppa:webupd8team/sublime-text-2
sudo apt-get update
sudo apt-get install sublime-text
```

For Sublime-Text-3:

```
sudo add-apt-repository ppa:webupd8team/sublime-text-3
sudo apt-get update
sudo apt-get install sublime-text-installer
```

Geany is also a nice, lightweight editor with a GUI.

Now, open an editor (I will prefer nano) and put down some aliases and every other stuff you want into this file. Finally, save this file as: **~/.bash_profile**

Append **source ~/.bash_profile** to the end of **.bashrc** file.

Try one of the aliases. Hmm, it does not work. May be I need a fresh terminal, or convince the current terminal to refresh the **.bashrc** file. Either open a new terminal, or on the current one try:

```
source ~/.bash_profile
```

or

```
source ~/.bashrc
```

What is the difference?

You made sure that from terminal to terminal your aliases will survive.

By the way, check the content of **~/.bash_history** for your amusement: **cat -n ~/.bash_history**

At this point if you are already in love with the terminal, and cannot keep your hands off of the keyboard, try vim, simply type: **vim**

Now on use the PING command. It is a usefull command to test you network status. Type:

```
ping google.com
```

Your screen should be flooded with package transmission statistics

Press **ctrl + z**

Ooops, where did the ping progress go? Answer: *suspended* at the background…

To convince yourself, list the current **p**rocesse**s** running in this terminal, simply type: **ps**

Ping process should be listed with a process ID (PID)

Now you can go to **b**ack**g**round by typing: **bg**

To get back to **f**ore**g**round try: **fg**

If you have multiple tasks in the **bg** & **fg**, at any point you can type **jobs** to see a list of tasks.

In case there is a process and you cannot get rid of it, or just for the fun of it, you can always kill a process.

Try **ps** to get a list of current processes. In case you are interested in processes beyond this current session of the terminal, try: **ps -e**

Any process that is to be sent to digital heavan, try: **kill PID**

where **PID** is the first argument in **ps -e** listing.

Remember that, if **ps -e** is too long to get what you are interested in, **grep** is in your service.

On a similar topic, if you are interested which processes are using a certain file (note that even hardware resources like serial ports are like files in Linux), you can try **fuser filename** to get the list of all processes using **filename**.

Now go to your **HOME** folder:

```
cd $HOME
```

or simply

```
cd ~
```

There we go, **~** stands for your $HOME, just to make sure try: echo $HOME

```
mkdir utils; cd utils
```

Source the following from a file for *almost* useless fun, give it the name **bg.test** first…

```
COUNTER=0
          while [  $COUNTER -lt 1000 ]; do
              echo The counter is $COUNTER
              sleep 1
              let COUNTER=COUNTER+1
          done
```

How will you get out of this sucker? Remember that **ctrl + c** will get you out of places where you do not want to stay!

Now check the content of the variable counter after breaking execution: **echo $COUNTER**

Well, now let's make this script executable:

```
chmod +x bg.test
```

and run this as: **./bg.test**

You can also create symbolic links to files, a.k.a. pseudonyms. Very helpful for generating alterative (preferably nice) names to files that you do not like the name but cannot also rename. Check out the command: **`ln source_file destination_file`**

For more info on chmod check out this wiki page

Now press **`ctrl + z`** to suspend execution.

Check the list of processes and jobs as well. To check jobs, simply type: **`jobs`**

Use bg, fg, ctrl + z to test the responses and also check the value of the variable COUNTER.

Observe the difference...

Also observe what is suspended when you source the script and run it as an executable.

Well the answer should be if you suspend the sourced script, most probably sleep command is suspended, but if you suspend the executable, the executable itself is suspended. You should have observed the difference on the screen!

Well, if all it takes to do usefull stuff is a terminal, how can I reach the terminal of some other machines?

If you know the name or the IP of the machine and you have a valid account on the machine you are good to go. Say you have a machine that is reachable on campus:

`ssh username@144.122.XXX.YYY`

Once you login, you have a remote terminal!

Just for fun try: **`who`**

This is just like the terminal on your local machine, but you are actually performing everything you do on a remote machine. For the fun of it try installing the scientific calculator and then run it: **`sudo apt-get install galculator`**

Read the message carefully :)

Type exit to exit and type: **`ssh -Y me461@144.122.49.132`**

Once logged in, type: **`galculator`**

Also try: **`arduino`**

If you attempt to do this from MAC you should have XQuartz installed at least. From Ubuntu / Mint etc, it should work right away.

#how to log info (time and IP) to a file – pay attention to the fact that some folders are hardcoded and you have to have a similar file structure to make it work... just go over the code and play with it.

```
pi<links_commands>$ cat log_time_ip
#!/bin/bash
#log current time and ip to along with passed parameters if any

logFile=~/startlog.log

echo "logging following to $logFile"
if test "$#" -ne 0
then
```

```
    #echo "custom massage passed"
    for i in "$@"
    do
        #echo "writing following to lo"
        echo "$i"
        echo "$i" >> $logFile
    done
else
    #echo "no parameters passed"
    echo "just logging" >> $logFile
fi

#current_ip=$(~/......./links_commands/get_ip)
echo $(sudo python3 ~/...../links_commands/get_ip)
echo $(date)
echo $(sudo python3 ~/...../links_commands/get_ip) >> $logFile
echo $(date) >> $logFile
echo "....." >> $logFile
echo "     " >> $logFile
echo "done..."
```

#to run this at startup on raspberry pi – (in case you have one and wonder how), append the following to **/etc/rc.local**

```
#log time and ip
python3 ~/...../links_commands/get_ip "fyi: just booted"
```

Recall that you can also schedule this operation for regular execution in crontab. To edit crontab try (carefully): **crontab -e**

But how on earth did we get the IP?

#how to get ip

```
pi<links_commands>$ cat get_ip
#!/usr/bin/env python
#this is python not BASH any more
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.connect(("gmail.com",80))
print(s.getsockname()[0])
s.close()
```

Try saving this and running it from python. And ask your self the question, without calling this function with python3, can I still execute it? Can I just call **get_ip** from command line? And the answer is YES. *You have to figure it out.*

At this point you deserve a break, how about watching Starwars like a nerd? ASCII Style, try:

```
telnet towel.blinkenlights.nl
```

Well, you know how to get **`telnet`** now. You do not need to keep it, remove it when you are done with the movie.

You should have been using your linux box for a while now, check: **`uptime`**
Talking about time, you can schedule tasks via **`crontab`**, and you can print a quick calendar on the terminal by just typing **`cal`**. By default current month is given yet you can get any year similar to: **`cal 2020`**

LONG LIVE THE TERMINAL

Couple of introductory tutorials on BASH programming,

and yet another one...

You can also try the commands reference on one of the most famous Linux Distros for Windows users, i.e., Linux Mint

next it is PYTHON time...