Robot Practical Course Bachelor
CheatSheet #1

This is the first Cheat Sheet. It is supposed to help you to understand some basic things about ROS and other things like how to use a terminal. We are also going to give you a basic understanding of how git works. This sheet is supposed to help you remember the things we have talked about in the beginning.

tldr;

Typical format of a shell command:

```
$ command -flags file1 file2
```

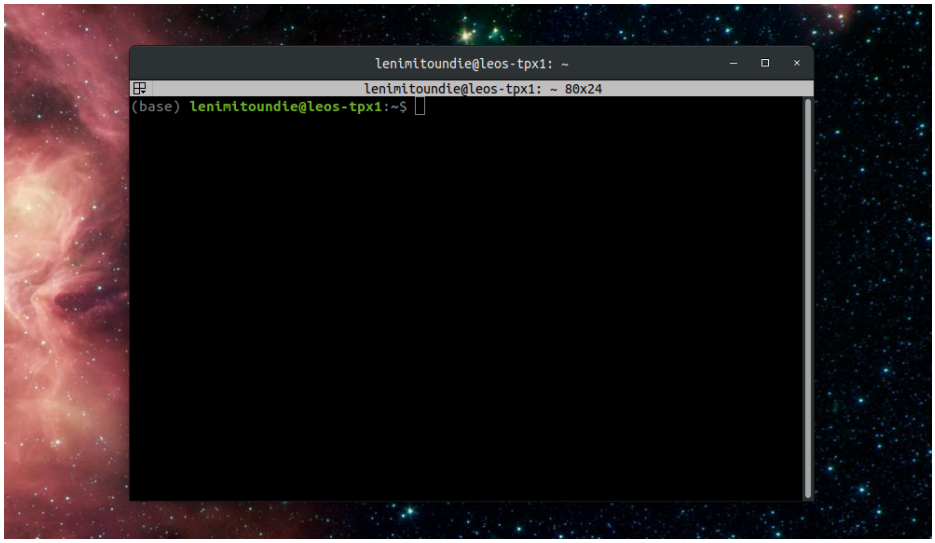
- flags: specify commands e.g. display all (also hidden) files l -a
- file1: often input file e.g. cp test.txt test-copy.txt
- file2: often destination/ output

Often used shell commands:

man <cmd>	opens manual for cmd (often also <cmd> -h)
cd	change directory
ll or l -al	lists all files and directories
chmod	allows to add or remove permissions
cp	copies things
rm	removes things PERMANENTLY
mv	moves things
pwd	path of working directory
mkdir	creates a new directory ("folder")
cat	displays content of a file (if possible)
touch	creates new file
clear	clears window - looks like newly opened

Topic 1.1 Bash: Most Linux Distributions as well as MacOS open as a standard login shell something called *bash*. It is a Unix Shell and a command language which can also read and execute commands from a file (shell scripts). If you have worked with a shell before you won't need this part of the sheet. Most programmers use a shell sooner or later in there career. As a CLI it is quite efficient and fast once you understand how to use it correctly.

On Ubuntu you can **open a terminal** (which connects to a console where the shell is presented) with **Strg + Alt + T**. Commands entered here will be executed by your operating system. If you've opened your terminal, you will see something like this:



It is important to understand that the shell is line-based, so if you hit enter, you execute your command. The **dollar sign $** you see at the end of the line is called **prompt**. After the prompt you enter your commands. What is displayed before the prompt varies, but it often gives you the main informations like *user, device-name* and it also shows you where in the file system you are. This is called the **working directory**. If you are not sure if you can see it, there is a command which displays it:

```
$ pwd
```

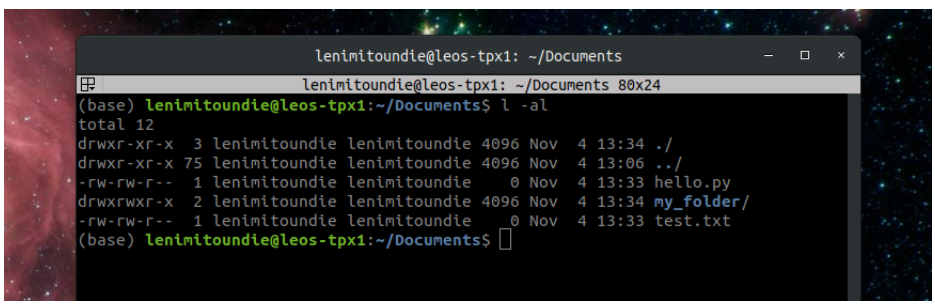
After executing it you will see something like

```
/home/user
```

Now that you know where you are, we will find out how to get somewhere else. If you want to change the directory you are in you use the command **cd**. Once you opened a new terminal, you will always start in your **home directory** which is represented as ∼. From there you might wanna move into your **Documents**/ directory. The command to do this is:

```
$ cd Documents
```

A file in your shell can be anything, it doesn't have to be a text file, every unit of something will be seen as a file. Images, Scripts, Text-Files are all files. As you would expect in your regular File-Explorer, you can move, copy, remove and duplicate files. You can change their name and also change who is allowed to read, write or execute them. Especially this last point is important when it comes to files that contain your programs. Per default your new file will only be allowed to be read and written too but it will **not be executable**! So you have to change this. But before we do that I want you to understand how you can see all files in your directory. We use the **l** command for that. With the flags **-al** it will display all files in your directory with additional information like the file size, a time stamp and the permissions we talked about before. They have a very simple format and look kind of like this:



As you can see, directories are slightly different than normal files. They don't have an suffix like *.txt* but end with a slash (e.g. my_folder/ in blue in the screenshot). They are often highlighted in another way as well. In this case bold and blue. The letters you can see in the beginning of the line shows you the permissions of the user, the group and others (which are neither the user nor in its group). In this order the permissions are displayed (first r is user, second is group and so on...). It looks something like this:

```
-rwxrw-r--
```

The **r** stands for a reading permission. The **w** for a writing permission. The **x** for a execution permission, which is quite important to have when the file contains code that you want to run.

You might have also noticed that sometimes this line starts with a **d**. It in indicates that this is a directory. Another thing you can see here are the directories that start with a period. Those are hidden directories that you normally can't see.

The **chmod** command allow you to change permissions. In the example underneath, *x* stands hereby for *executable* and the plus sign simply means that you want to add this permission rather than remove it.

```
$ chmod +x <filename>
```

Topic 1.2 ROS explained: ROS 2 is not an operating system and more like a framework that helps to abstract and reuse code between different platforms. There are different concepts you have to understand to work with ROS:

- **Nodes**: Nodes are processes that perform computation. ROS is designed to be modular at a fine-grained scale; a robot control system usually comprises many nodes. For example, one node controls a laser range-finder, one node controls the wheel motors, one node performs localization, one node performs path planning, one Node provides a graphical view of the system, and so on. A ROS node is written with the use of a ROS client library, such as rclcpp or rclpy.
- **Topics**: Messages are routed via a transport system with publish / subscribe semantics. A node sends out a message by publishing it to a given topic. The topic is a name that is used to identify the content of the message. A node that is interested in a certain kind of data will subscribe to the appropriate topic. There may be multiple concurrent publishers and subscribers for a single topic, and a single node may publish and/or subscribe to multiple topics. In general, publishers and subscribers are not aware of each others' existence. The idea is to decouple the production of information from its consumption. Logically, one can think of a topic as a strongly typed message bus. Each bus has a name, and anyone can connect to the bus to send or receive messages as long as they are the right type.
- **Services**: The publish / subscribe model is a very flexible communication paradigm, but its many-to-many, one-way transport is not appropriate for request / reply interactions, which are often required in a distributed system. Request / reply is done via services, which are defined by a pair of message structures: one for the request and one for the reply. A providing node offers a service under a name and a client uses the service by sending the request message and awaiting the reply. ROS client libraries generally present this interaction to the programmer as if it were a remote procedure call. Calling a service is a blocking operation. Due to internals in ROS2 this blocks the executing thread and leads to a deadlock as the

answer callback can not be executed if we wait in the same thread. To avoid this, use done callbacks that get executed when the answer arrives or use an async python function and await the answer.

- **Messages**: Nodes communicate with each other by passing messages. A message is simply a data structure, comprising typed fields. Standard primitive types (integer, floating point, boolean, etc.) are supported, as are arrays of primitive types. Messages can include arbitrarily nested structures and arrays (much like C structs).
- **Launch Files**: Launch files are XML files that define the configuration of a ROS system. They specify which nodes to launch, their parameters, and any necessary remappings of topics. Launch files allow for easy orchestration of complex robot systems by providing a single entry point for starting all necessary components.