Doing the following exercises prepares you for using Linux, Git and ROS.

> **Disclaimer**:
>
> - **The tasks usually require a substantial amount of work, please start as early as possible**
> - **If you need help, you can google, or ask on the class forum, you can also attend the labs in person and ask a question directly to the teaching assistants**
> - **Please don't forget to give us feedback on this task.**

Overview of the task

Complete the tasks 1.1 to 1.5. You will need this knowledge and the mandatory tools to continue with the course, so it's important to be finished with these exercises before next week's lab.
For the tasks, you will mostly follow a step-by-step guide where all required actions are given to you in detail. Only sometimes, you are asked to apply previously acquired knowledge or to find out by yourself how to accomplish a task.
There is no submission required for the introductory lab tasks. But since they form a basis for the knowledge you need when approaching the other lab tasks, we highly encourage you to be thorough nonetheless.
In this course we will refer to ROS or ROS2 interchangeably, in all cases this refers to ROS2.

Task 1.1: Installing Ubuntu

ROS needs Ubuntu or Debian Linux and some parts of it don't run well on virtual machines.

- Install Ubuntu 22.04 Jammy. For the course we recommend adding a partition on the hard drive of your computer (dual boot), install Ubuntu 22.04 Jammy , and use the dual-boot option for booting into Ubuntu environment. You can find many good tutorials on the net about how to dual boot (one example here). Windows subsystem for linux, booting from a USB drive, or Virtual machines are also options but be warned: we do not recommend these options. Again, we recommend dual boot.

Task 1.2: Practice linux

Follow this short tutorial to get acquainted to the most important Linux commands. You will mostly use "Terminal" to interface with your operating system and your files. To be able to do so, it is necessary to have the most common commands in mind. If you feel you would like to visit a more official tutorial guiding you through the Linux commandline interface, visit this website. If not, let's get started with this short introduction.

First off, open a Terminal window (CTRL +ALT + T) .

ls (list)

When you first login, your current working directory is your home directory. It has the same name as your username, and it is the place where your personal files and subdirectories are saved. To find out what is in your home directory, type ls
The "ls" command (meaning "list") lists the contents of your current working directory. It does not, in fact, cause all the files in your home directory to be listed, but only those ones whose name does not begin with a dot (".") because files beginning with a dot (".") are known as hidden files and usually contain important program configuration information. They are hidden because you should not change them unless you are very familiar with Linux!

To list all files in your home directory including hidden files, type

ls -a (list all)

As you can see, "ls -a" lists files that are normally hidden. "ls" is an example of a command which can take options: "-a" is an example of an option. The options change the behaviour of the command. There are online manual pages that tell you which options a particular command can take, and how each option modifies the behaviour of the command. Can you understand what changes when you type:

ls –al (list all long)

mkdir (make directory)

We will now make a subdirectory in your home directory to hold the files you will be creating and using in the course of this tutorial. To make a subdirectory called "mystuff" in your current working directory type

mkdir mystuff

To see the directory you have just created, type

ls or ls -la

cd (change directory)

The command "cd directory_path" would mean to change the current working directory to "directory_path". The current working directory may be

thought of as the directory you are in, i.e. your current position in the file-system tree.

To enter the directory you have just made, type

`cd mystuff`

Now list the directory content. Use your existing knowledge! The directory should be empty.
Make another directory inside "mystuff" which is called "backups".
Now list the content of "mystuff" again. Can you see the newly created directory?
Please note that there are two additional directories present. One is "." and the other "..".

The current directory (".")

In Unix, "." means the current directory. Typing `cd .` instructs the operating system to navigate to the place where you currently are. As a result, you stay where you are. Whenever you specify a path to a file or folder in the terminal, the "." in the beginning of the path specifies that this path is a relative path, given from the point of the current directories. In other words, paths starting with "." depend on the current directory where the user is located when (s)he types the path. Using "." to specify a relative path may save a lot of typing, as we will see later in this tutorial.

`cd ..` will take you one directory up the hierarchy (to the parent directory). Try it now.
Note: Typing `cd` with no argument always brings you back to your home directory, and is the same as `cd ~` or `cd /home/<username>`. This is useful if you're lost in the file system.

pwd (print working directory)

Pathnames enable you to work out where you are in relation to the whole file-system. For example, to find out the absolute pathname of your current directory, type `pwd`

Understanding path names

First go back to your home directory. Use existing knowledge to do that.

Then type `ls mystuff` to list the contents of the "mystuff" directory.
Now type `ls backups`

You will get a message like this:
backups: No such file or directory

This happens because "backups" is not a directory inside your current working directory (the home directory in this case). You are able to list the content of "mystuff" using "ls mystuff" because "mystuff" is indeed a subdirectory of the current working directory.
In short, if you type just "ls", it lists the content of the current directory

(wherever you are at the moment). If you type "ls subdir", it lists the content of a directory "subdir" which is inside the current working directory.
If you want to list the content of a subdirectory "subsub" of a subdirectory "sub" of the current working directory, you must either "cd" into "sub" and then call "ls subsub", or you stay in the current working directory and type "ls sub/subsub"

To list the content of "backups" without changing the working directory, you can type

`ls mystuff/backups`

Your home directory ("~")

The home directory can be explicitly specified by the tilde ("~") character. It also specifies paths starting at the home directory.

`ls ~/mystuff` lists the content of "mystuff", no matter where you currently are in the file system. This being said, it becomes obvious that the path you just specified here is not relative, but absolute. The "special thing" about this path is that it uses "~" as a substitute for the absolute path to the home directory: /home/<username>.

What do you think `ls ~` would list?

What do you think `ls ~/..` would list?

Let's recap!

Command	Meaning
ls	list files and directories
ls -a	list all files and directories
mkdir	make a directory
cd subdir	change into directory "subdir"
cd	change to home directory
cd ~	change to home directory
cd ..	change to parent directory
pwd	display the path of the current directory

touch (create a file)

Let's create a new file in your home directory. Change into your home directory (if you haven't yet) and type
`touch test.txt`
Now use existing knowledge to list the content of your home directory. Do you see the newly created file?

cp (copy)

"cp file1 file2" is the command which creates a copy of "file1" and places this copy under the name "file2". "file1" and "file2" can be file names or paths to files.
You are now going to take a file stored in the home directory and use the "cp" command to copy it into the "mystuff" directory.
Change into "mystuff" so it becomes your current working directory:
`cd ~/mystuff`
Then type
`cp ~/test.txt .`
Do not forget the "." in the end. It is the second argument of the "cp" command and tells to the system what should be the place of the copy to be created. As you already know, the "." specifies the current working directory, in this case "~/mystuff". You might notice that we did not specify a file name for the file to be copied, just the path (".") . Whenever you want to create a copy of a file with the same name as the original (but in another directory), you don't need to specify the target file name, only the path.
Using your new knowledge, create a backup of "test.txt" by copying it to a file called "test.bak" in the same directory as the original. List the content of the directory to assert the file is there.

mv (move)

"mv file1 file2" is the command which moves a "file1" into "file2". "file1" and "file2" can be file names or paths to files. If the path to file1 and the path to file2 are equal, but only the file names are different, file1 will be renamed into file2.
In contrast to "cp", "mv" leaves you with only one copy of the file, but in a new location or under a new name.
You are now going to move the file "test.bak" to your backup directory.
First, change the current working directory to "~/mystuff".
Using "mkdir", create a new directory named "backups" inside of "mystuff".
Then type
`mv test.bak backups/.`
Type `ls` and `ls backups` to see if it worked.

rm (remove), rmdir (remove directory)

To delete (remove) a file, use the "rm" command. As an example, we are going to create a copy of "test.txt" and then delete it.
Inside your "mystuff" directory, type

```
cp test.txt tempfile.txt
ls
rm tempfile.txt
ls
```

You can use the "rmdir" command to remove a directory (make sure it is empty first).
Try to remove the "backups" directory. You will not be able to do so since Linux will not allow you to remove a non-empty directory.
Create a directory called "tempstuff" using "mkdir", then remove it using the "rmdir" command.

clear

Before starting the next section, you may want to clear the content of your terminal window.
At the prompt, type
`clear`
Now please download the file "license.txt" and save it in the home directory.

cat (concatenate)

The command "cat" can be used to display the contents of a file on the screen. Type
`cat license.txt`
As you can see, the file content does not fit into the window all at once, so it scrolls past - which makes it unreadable.

less

The command "less" writes the contents of a file onto the screen one page at a time. Type
`less license.txt`
Then, still in "less", type a forward slash ("/") followed by a word to search for: /source
Then hit return. "less" finds and highlights the keyword.
If you just want to scroll (no search), hit the space bar. To exit, type "q".

grep (globally search for a regular expression and print matching lines)

"grep" is one of many standard Unix utilities. It searches files for specified words or patterns.
First clear the screen. Then type
`grep source license.txt`
Apparently, "grep" has printed each line containing the word "source".
But now, type `grep Source license.txt`
The "grep" command is case-sensitive; it distinguishes between capitalized and non-capitalized letters.
To ignore uppercase and lowercase distinction, use the "-i" option:
`grep -i source license.txt`

To search for a phrase or pattern, you must enclose it in single quotes (the apostrophe symbol). For example, to search for "terms and conditions", type
`grep -i 'terms and conditions' license.txt`
Some of the other options of grep are:

- "-v": display those lines that do NOT match
- "-n": precede each matching line with the line number
- "-c": print only the total count of matched lines

Try some of them and see the different results. Be aware that you can use more than one option at a time. For example, the number of lines without the word "source" and "Source" is|
`grep -ivc source license.txt`
The following adds line numbers to the printout of the search results for "terms and conditions":
`grep -in 'terms and conditions' license.txt`

Let's recap!

Command	Meaning
touch file	create new file named "file"
cp file1 file2	copy "file" and name it "file2"
mv file1 file2	move or rename file1 to file2
rm filename	remove a file
rmdir directoryname	remove a directory
cat filename	display a file
less filename	display a file one page at a time
grep 'search for' filename	search a file for keywords

Task 1.3: Use git and cloud storage (Optional and recommended)

For this course, you will have to write programs and create a lot of files. It is your own responsibility not to lose these files and to keep track of iterative versions of your files. "Git" is a version-control system that allows programmers to track changes in their development and store their results safely in the cloud.
To prevent your data from getting lost and in order to manage versioning of your files, you are encouraged to use Git. This assignment contains a short introduction. If you prefer a nice online tutorial with plenty of examples and puzzles, click here. Otherwise, follow this short introduction (which won't cover as much, but may give you a very basic understanding nonetheless).

Git setup

You can use Github or Gitlab for managing your repository. Git is a version control system that tracks changes in code, ensuring data integrity and preventing data loss. GitHub and GitLab are web-based platforms that host Git repositories, providing online backup and access to your repositories.
On github, create a user account (if you don't have yet) or use your university's gitlab account: https://gitlab.cs.ttu.ee/users/sign_in
Create a new repository using the web interface and name it "ias0220" (for example).
Find out what the web address of your new repository is.
Hint: On the github website, find the "Code" button and click on it (see image below).

Open the Linux terminal and go to your home directory.
Then type
`git clone <url>` where "<url>" is the link to your repository. Use "ls" to confirm that the repository has been downloaded into your home directory.
In terminal, navigate into your git directory.
Configure git with your information so that remote repository (for example gitlab) would know who is sending these changes. For that type

`git config --global user.email "youremail@email.com"` - to configure git with your email (use your gitlab email).

`git config --global user.name "Your Name"` - to configure git with your name.

Usual workflow with Git

First thing is to make some changes in your code. For example type
`gedit` to open a text editor. We recommend `code` (Visual studio Code), that you'll have to install first. Write something into the file and save it inside your local git directory.

Recommended filename for repository documentation is "README.md".
Close gedit.
Type `git status`
You should now see the newly created text file in red. It means that this file has changed since the last commit of this repository, of course: The file did not even exist before.
Using the same file name, type
`git add filename_in_red`
Doing so, you add this change to the "stage" - meaning that it will be ready for being committed to the online repository, from where all other collaborators can pull (= download) it next time they log in. Another possibility is to use "git add ." command which adds all uncommited files to be commited in your working directory.
However, the file is just staged and not yet committed.
Again type
`git status`
You should see the same file name, but in green. It means the file still does not exist in the current version of the repository, is not yet committed, but is already staged for being committed.
If you now type
`git commit -m "Learning to use git"`
you tell to git that all files that are staged at this moment can be uploaded to the online repository next time you "push" your repository from your local hard drive to the cloud. The "-m" flag is followed by a message which describes the reason of this commit or the most important changes in it. Everyone who downloads the repository can see these commit messages.
Finally, let's "push" (upload) the committed files to the online repository so other collaborators can see the changes:
`git push`
In your web browser, check on the git website that the upload has been successful.

Git is also very useful for teamwork. It is possible to create branches of the same code, do different development on that code simultaneously between team members and merge everything back together. Branches are also useful to **save different states of your code** or **when you want to test something out without loosing currently working code**. Even though it is very useful and way more effective than doing full backup of your code, merging branches together may be difficult at the beginning and therefore we suggest you to stay in "master" branch and not to do branching before you have learned more about git (from here).

Task 1.4: Getting started with ROS

To get a good overview and understanding of ROS2, please open the tutorial page.
During this course, you will complete a large part of the ROS beginners' level tutorial.
These tutorials walk you through installing ROS and setting up the ROS environment on your computer. Although if you are using the USB image, you have ROS already installed as part of the image we provided, you will need to know how to build a

"workspace", so ROS can run the programs you write.
For the next lab session, please complete the following sections:

1. Installing ROS2. Install ROS2 HUMBLE
2. Configure your environment
3. Using colcon to build packages **NB!** When you define a path for your workspace, make sure it does not contain any spaces!

   Now you should install git. If you haven't already, you can install it with: `sudo apt install git-all`
   Also install and initialize rosdep. It is a ROS utility that identifies and installs dependencies to build or install a package:
   `sudo apt install -y python3-rosdep2`
   `sudo rosdep init`
   `rosdep update`
4. Create a workspace
   This tutorial shows you how to create a workspace, clone a repo from git, resolve dependencies, build and modify the package you just cloned in your overlay
5. Creating a ROS2 Package
   This tutorial shows you how to create and build your first package and customize the mandatory files

> **Note**: if when you build packages you have the following warning in your terminal: "SetuptoolsDeprecationWarning: setup.py install is deprecated. Use build and pip and other standards-based tools.", you can ignore it, or you can downgrade setuptools:
> 1 - verify that you have setuptools 59.6.0 or higher: `pip3 list | grep setuptools`
> 2- downgrade to 58.2.0: `pip3 install setuptools==58.2.0`

Task 1.5: Install supplementary software

Additional software required:

1. imu_tools for ROS

   This is required in order to be able to display IMU messages in Rviz using the IMU object.
   You will need this for at least one exercise at a later time in this course.
   `sudo apt install ros-humble-imu-tools`

   Docu: https://github.com/CCNYRoboticsLab/imu_tools/tree/humble

2. tf2 for Python

This library offers a bunch of methods to transform between various mathematical representations of a robot's pose, among others.
You will need this for at least one exercise at a later time in this course.
`sudo apt update`
`sudo apt-get install ros-humble-turtle-tf2-py ros-humble-tf2-tools ros-humble-tf-transformations`

3. pip for python3

Some of the tools you will need can be installed through pip. To install it you can type
`sudo apt install python3-pip`

Additional software which might be useful (but not required):

Visual Studio Code

This software offers good code editors for all common programming languages. It is a lightweight version of Microsoft*s Visual Studio and runs on Ubuntu.
You can install it from the ubuntu software app

Tree

This software enables you to see the tree structure of a directory in the terminal
`sudo apt install tree`

Summary:

In this first task, you set up your computer by installing Ubuntu (Linux), and started to learn linux command line tools. You then installed ROS2 Humble and built your workspace, your first package and learned the basics of ROS2 packages filesystem and files functions. But let's get a taste of robotics already! In the next task, you will build your robot in the Unified Robot Description Format (URDF)!