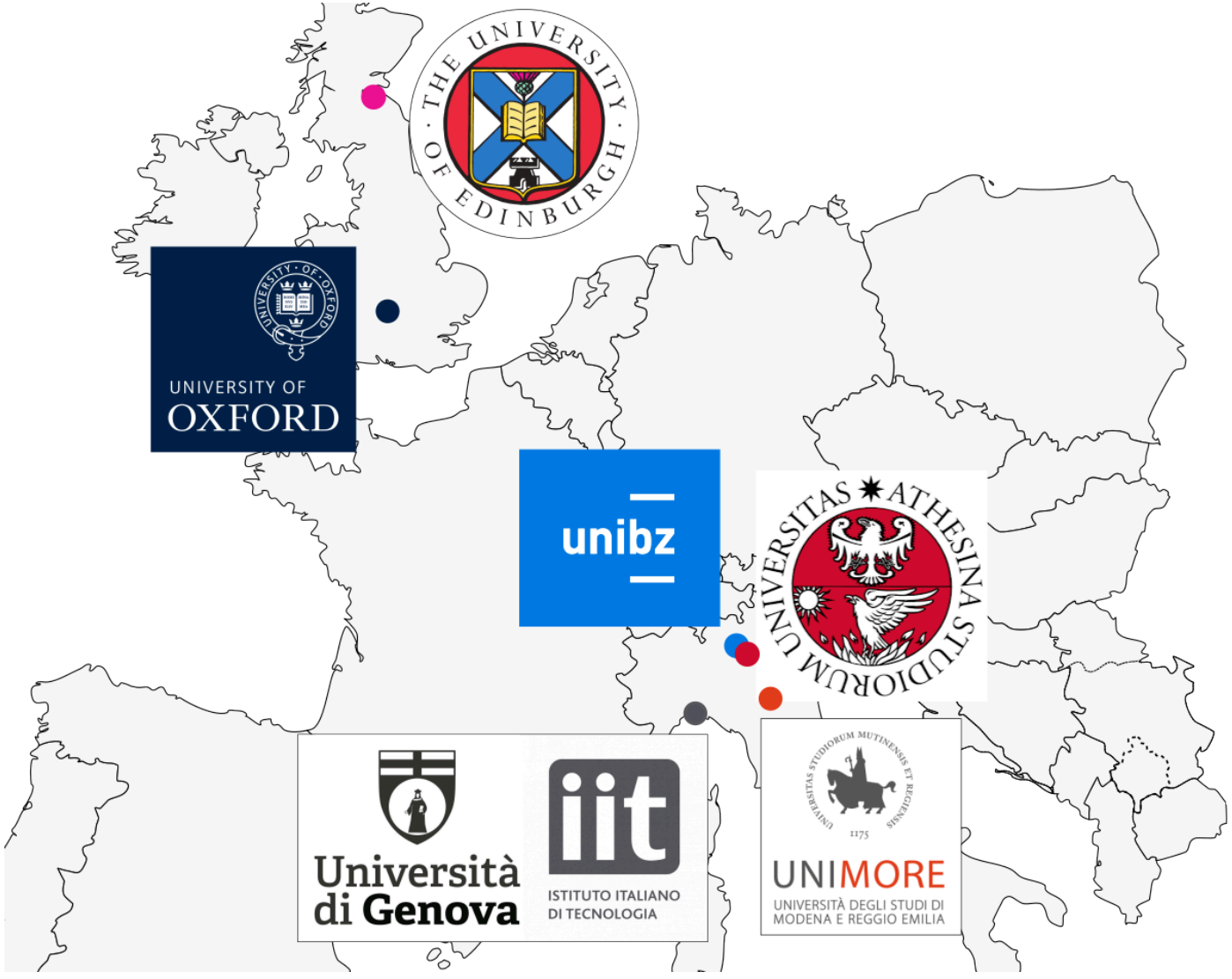
Introduction to Robotics

LAB 01 - Introduction to ROS

Marco Camurri

About Me



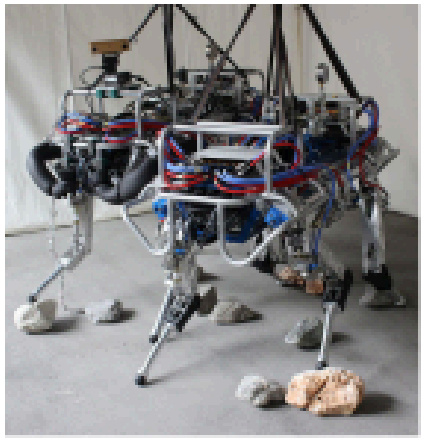
2024-now	Associate Professor
2023-2024	Assistant Professor
2021-2022	Senior Researcher
2018-2020	Post-Doc
2017-2018	Post-Doc
2015	Internship
2014-2017	PhD Avanced Rob.
2013	Research Assist.
2009-2012	M.Eng Comp. Vision
2006-2009	B.Eng Comp. Science

Lots of field trials...






… with really cool robots!

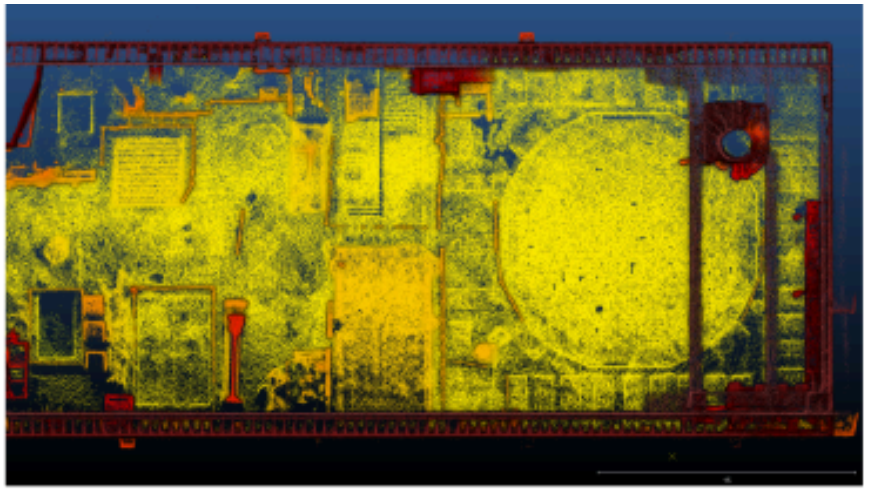




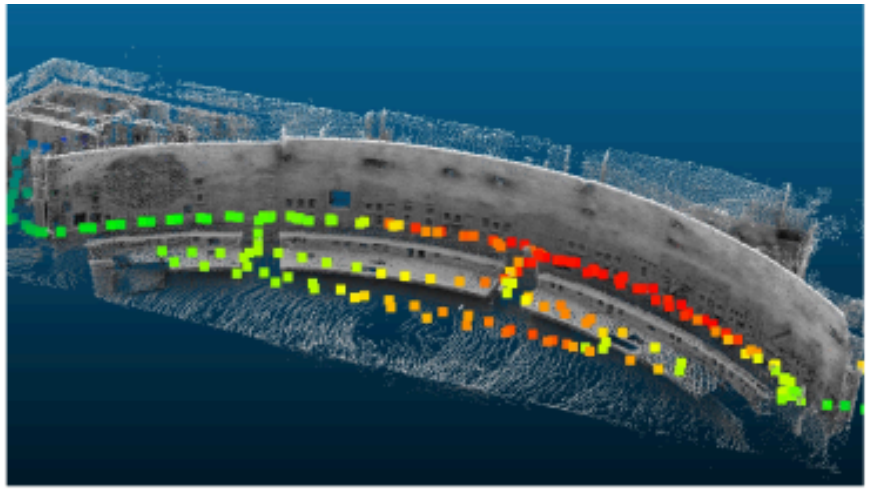


Mapping the Chernobyl Nuclear Power Plant

- **VILENS** as motion prior
- **LIDAR SLAM** for loop closure detection and fusion
- **4 radiation sensors** for radiation mapping



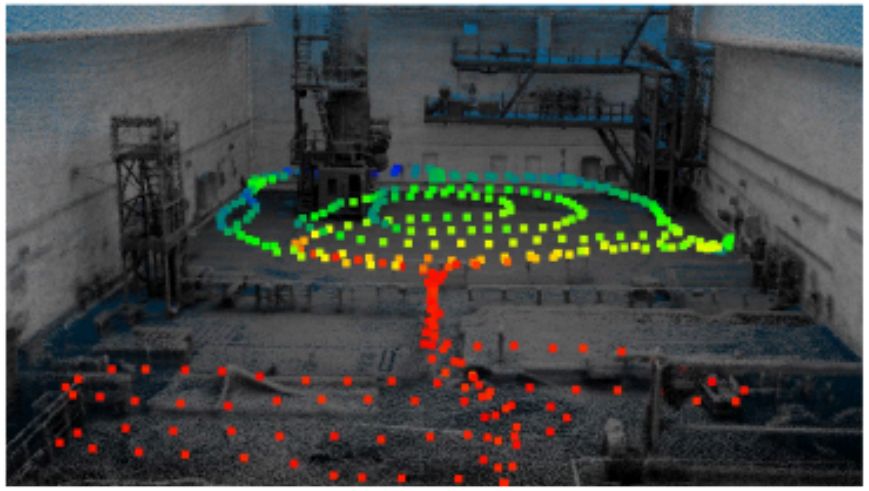
Reactor 3 Hall Top view



Reactor 4 Control Room (Mapping and Radiation Scan)

RAIN

University of BRISTOL



Reactor 3 Hall (Mapping and Radiation Scan)

Reactor 3 Hall (decommissioned)

Not sure what to do for your Thesis?

Let's stay in touch!

`marco.camurri@unitn.it`

On the menu today …

- Ubuntu Command-Line Basics
  - Terminal
  - Shell commands
- Introduction to ROS
  - What is ROS
  - ROS Concepts: Packages, Topics, Nodes
  - ROS Tools: TF, RViz, Gazebo
- Interactive Session: install ROS and use some basic commands

Ubuntu Command-Line Basics

The terminal

- The only way to interact with a computer before Graphical User Interfaces (GUIs)
- Some robots don't have a GUI, you need to know how to use a Terminal!
- Modern Operating Systems have a **Terminal Emulator**
- On Ubuntu you can open one pressing `Ctrl + Alt + T`

File system commmands: `pwd`, `ls`, `cd`, `man`

- `pwd` stans for *print working directory* and ... prints the working directory
- `ls <folder_name>` stands for *list* and shows the content of a folder. To list the content from the current folder you can run with no arguments
- `cd <dir_name>` stands for *change directory* and changes the working directory. Two special directories are the current `.` and parent `..`
- `man <command_name>` (stands for *manual*) don't know what a command does or how to use it? Just use this command to know more about it. If it says `No manual entry for ...` then no manual page is available, but you can try to run the command followed by `-h` or `--help`

Create new files and folders: `mkdir`, `touch`, `vim` and `nano`

- `mkdir <dir_name>` stads for *make directory*, makes an empty directory with a given name
- `touch <file_name>` updates the time of a file. If the file doesn't exist, it creates an empty one
- Two common text editors are `vim` and `nano`. The first one is more advanced, the second is more beginner friendly.

Copy, move and remove files: `cp`, `mv`, `rm`

- `cp <file_name> <dir_name>` copies a file in the selected directory. If a file name is given instead, the copy has that name
- `mv <file_name> <dir_name>` moves a file to a directory. If a file name is given instead, the file changes to that name. ❗ Trick to rename a file: just move a file in the same place where it already is.
- `rm <file_name>` erases a file from disk. **WARNING ❗ ❗ ❗ this is a non recoverable action!** If you do `rm -rf <dir_name>` it erases an entire directory and all subdirectories with no mercy.

Installing new software: `apt`

To install a package:

```
sudo apt install <package-name>
```

where `sudo` gives you superpowers (it stands for "*super user do*"), `apt` is a command (a package manager) and `install` is a self-explanatory argument for the `apt` command
Example, to install ROS you ran:

```
sudo apt install ros-noetic-desktop-full
```

Other useful commands

- `cat <file_name>` shows the content of a file by printing it on screen.
- `wget <url_name>` downloads a file from a given URL
- `tar -xvf <file_name>` decompresses a zipped file. Other arguments can be used to compress files but we won't use them very much
- `echo <something>` Will print something.
- `find <dir_name> -name <regex>` will recursively find all files or folders matching a regular expression inside a directory. Example: `find . -name *.pdf` will find all PDF files in the current folder
- `grep <regex> <dir_name>` is an extremely powerful tool to find strings in files or folders. Sometimes you'll see something like: `grep -r *robot* .`

Nice tricks: pipe and redirection

- `|` this is the pipe symbol. It is used to plug the output of a command to the input of another command. Try the following:

```
sudo apt install fortune cowsay
fortune
cowsay hello
fortune | cowsay
```

- `>` this is the redirection symbol. It redirects the output of a command from standard output to a file. For example: `echo Hello > hello.txt` will create a file called `hello.txt` with inside the string `Hello`.

Version Control System: `git`

- From the official website:

> Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

- Git is a fundamental tool to develop code efficiently in teams while keeping track of all of the changes.
- We won't cover `git` in this course specifically, but you might hear the terms *clone* or *git repository* from time time time.
- You should learn `git`. Read this book. And try it: `sudo apt install git`

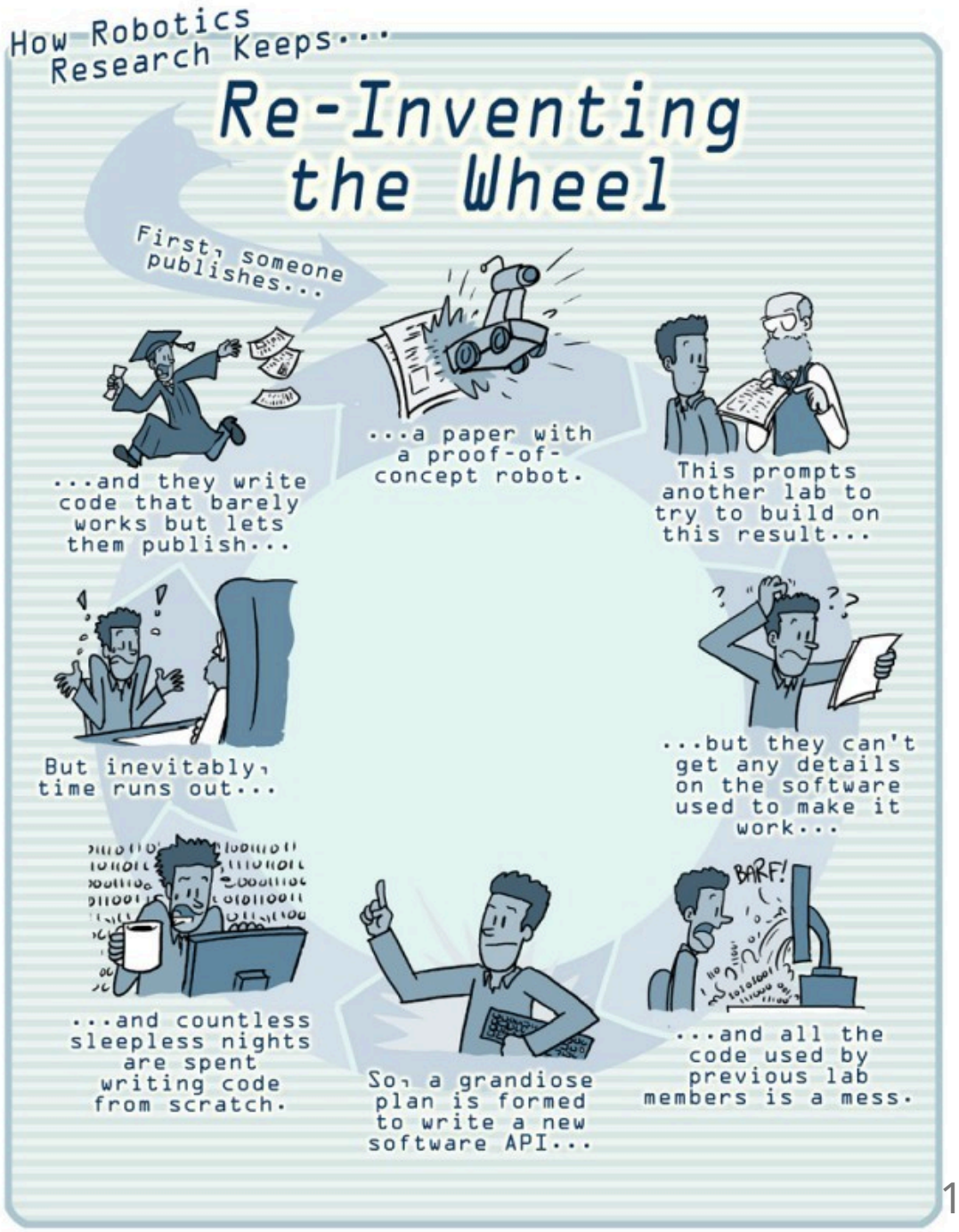
The `.bashrc` file

- On linux systems, files with a name that begins with `.` are hidden
- In your home folder is located a hidden file called `.bashrc`
- Any definition (commands, aliases, variables) stored in there will be available once you open a terminal
- We will alter our `.bashrc`, mostly to updated the *environment variables*
- An example of environment variable is `PATH`. Try printing it: `echo $PATH`

Introducting the Robot Operating System (ROS)

What is ROS ?

- **ROS is NOT an Operating System** 😅
- It is a **collection of programs and libraries** to facilitate robot programming.
- Some people would define it as a *middleware*.



Is ROS better than alternatives?

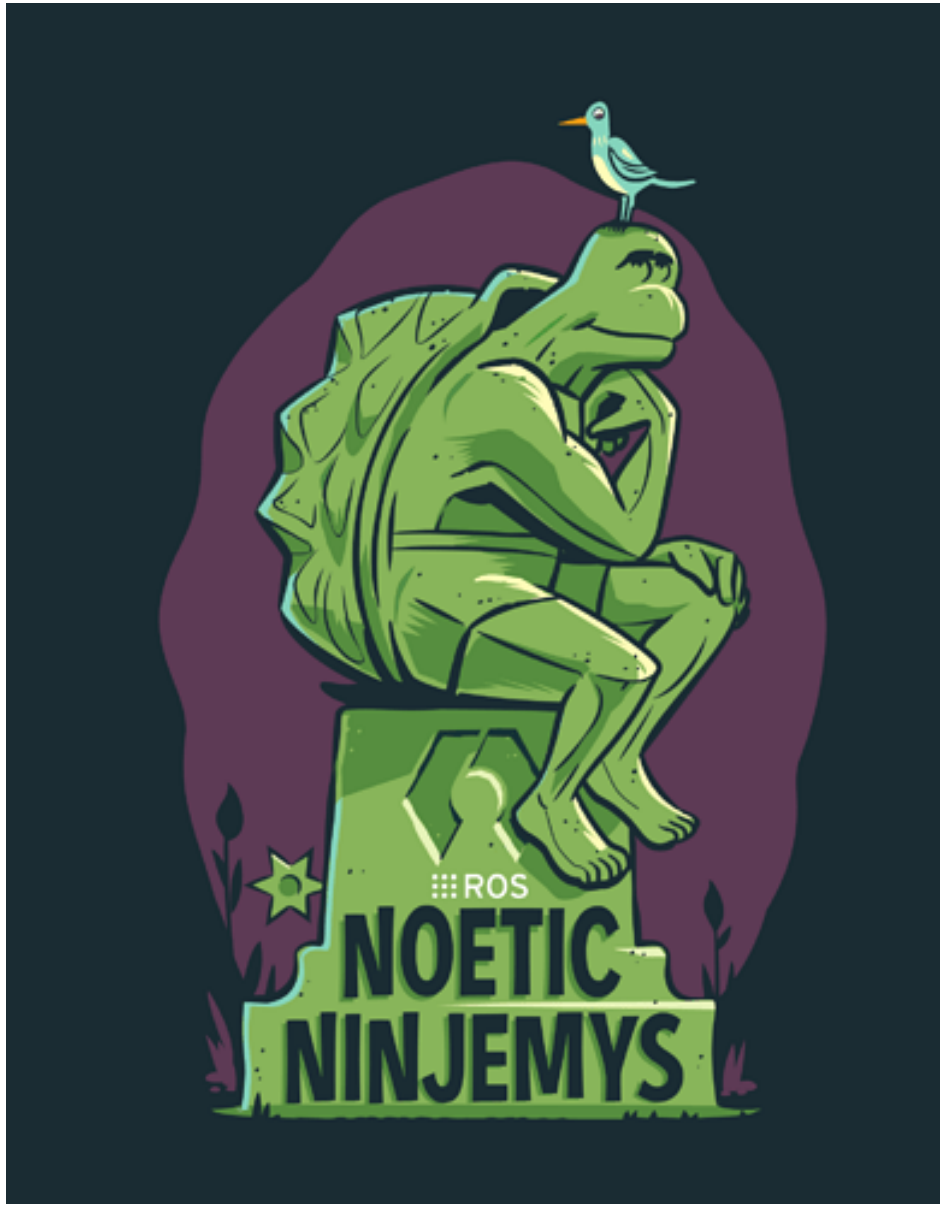
> No. Quicker. Easier. More seductive.

ROS is a collective effort to stop reinventing the wheel by creating a large community of researchers and hobbyists who SHARE code, programs, configurations, models, simulation, and data.

This is what makes it a *de facto* standard for robotics, like TCP/IP for networking.

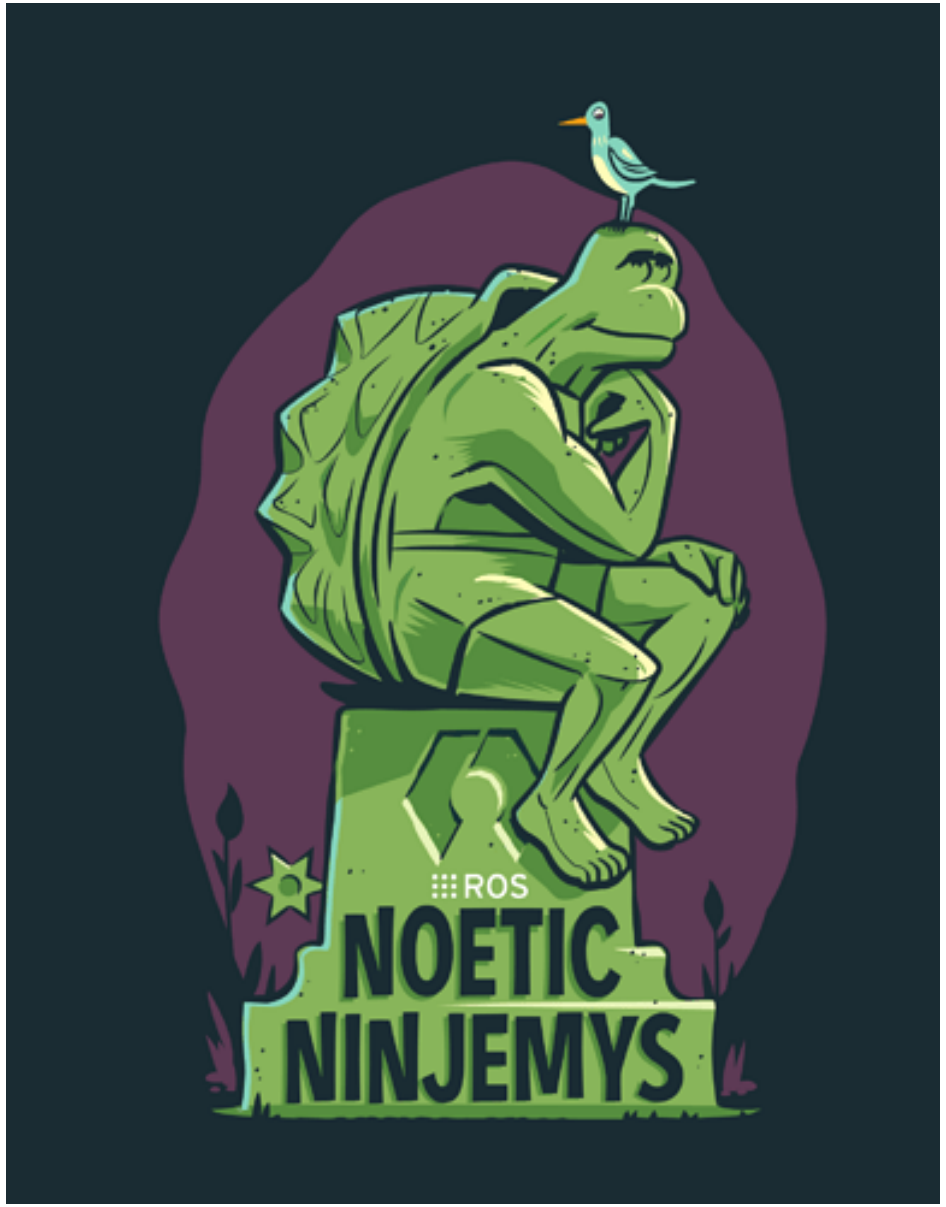
ROS Releases

- ROS stated in 2007 and its mascotte is a turtle.
- There are typically **two new releases per year**
- Every two years, in May, there is a new **Long Term Support (LTS)** release
- Each release is coupled with a specific version of Ubuntu



Release names

- Each release's name is an adjective followed by the name of a turtle species
- The release taught in this course is Noetic Ninjemys
- It needs to be installed on Ubuntu 20.04
- This will be the last version of ROS 1 and will be supported until May 2025 😕



ROS 2

- **ROS 2 exists** for a few years already, but **it is more complicated**, less mature, and less documented than ROS 1
- **For simplicity, we use ROS 1 in this course**
- Learning ROS 2 when you already know ROS 1 is not difficult
- When appropriate, we will point out the differences between the two systems

What does ROS offer to you?

- A ready to use library for distributed Inter-process communication, message generation, marshalling, and unmarshalling
- A wide range of software libraries and programs for robot simulation, control, perception, navigation, optimization,
- A standardised packaging system, to ease the development of new libraries

- Extensive library of ROS wrappers for most commonly used sensors drivers (most of them provided by the manufacturer) Robot models
- A big community for wiki, tutorials, support, etc.
- Server farms where your libraries can be shared amongst other users, which would just have to install and use them

Knowledge of ROS is an essential skill for a robotics engineer.

Disclaimer

The following material is partially adapted from the following courses:

- "Programming for Robotics" from ETH Zurich. https://rsl.ethz.ch/education-students/lectures/ros.html
- "Visual Navigation for Autonomous Vehicles" from MIT https://vnav.mit.edu/

ROS Philosophy

- **Peer to peer**: individual programs communicate over defined API (ROS *messages*, *services*, etc.)
- **Distributed**: programs can be run on multiple computers and communicate over the network
- **Multi-lingual** ROS modules can be written in any language for which a client library exists (C++, Python, MATLAB, Java, etc.).
- **Light-weight**: Stand-alone libraries are wrapped around with a thin ROS layer.
- **Free and open-source**: Most ROS software is open-source and free to use.

ROS Master

- ROS 1 uses a **centralised way of communication**
- The **ROS master** manages the communication between nodes
- Nodes are regular programs that communicate with each other using ROS (more in next slide)
- Every node registers at start up with the master
- To start a master, just type and run `roscore` on your terminal. You can only run one ROS master at the time
- **NOTE**: In ROS 2 the communication is no more centralised. Nodes discover and talk to each other directly, which makes the protocol more complex

ROS Nodes

- Single purpose, executable programs
- Individually compiled, executed, and managed
- Organised in packages (see later slides)
- Each package can have multiple nodes

Run node:

```
rosrun package_name node_name
```

See active nodes:

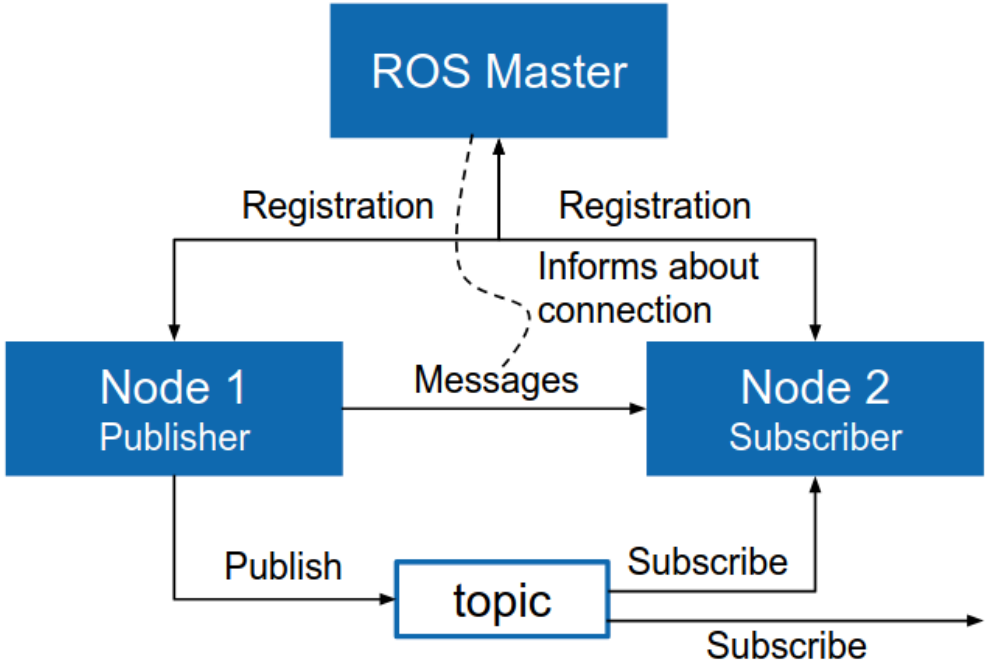
```
rosnode list
```

Retrieve information about a node:

```
rosnode info node_name
```

ROS Topics

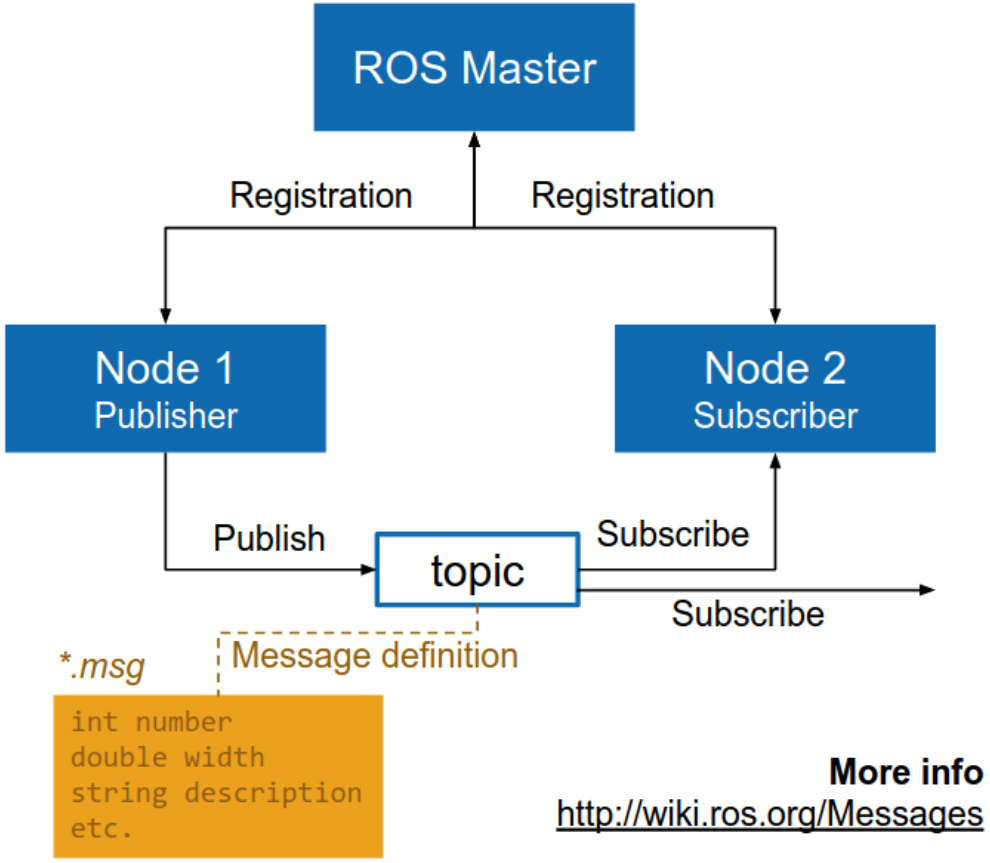
- Nodes communicate over topics
- Nodes can *publish* or *subscribe* to a topic
- Typically, 1 publisher and n subscribers
- `topic` is a name for a stream of *messages*
- List active topics: `rostopic list`
- Subscribe and print content on topic:
  `rostopic echo /topic`
- Show information about a topic:
  `rostopic info /topic`



ROS Messages

Data structure defining the *type* of a topic

- Comprised of a nested structure of integers, floats, booleans, strings etc. and arrays of objects
- Defined in `*.msg` files
- See the type of a topic: `rostopic type /topic`
- Publish a message on a topic: `rostopic pub /topic type data`



ROS Message example `PoseStamped`

- geometry_msgs/PoseStamped.msg defines the message with two fields:

```
std_msgs/Header header
geometry_msgs/Pose pose
```

- std_msgs/Header.msg contains the definition of a `Header`:

```
uint32 seq
time stamp
string frame_id
```

- geometry_msgs/Pose.msg is defined as:

```
geometry_msgs/Point position
geometry_msgs/Quaternion orientation
```

- geometry_msgs/Point.msg is defined by the point coordinates:

```
float64 x
float64 y
float64 z
```

- geometry_msgs/Quaternion.msg is defined by the fields of the quaternion

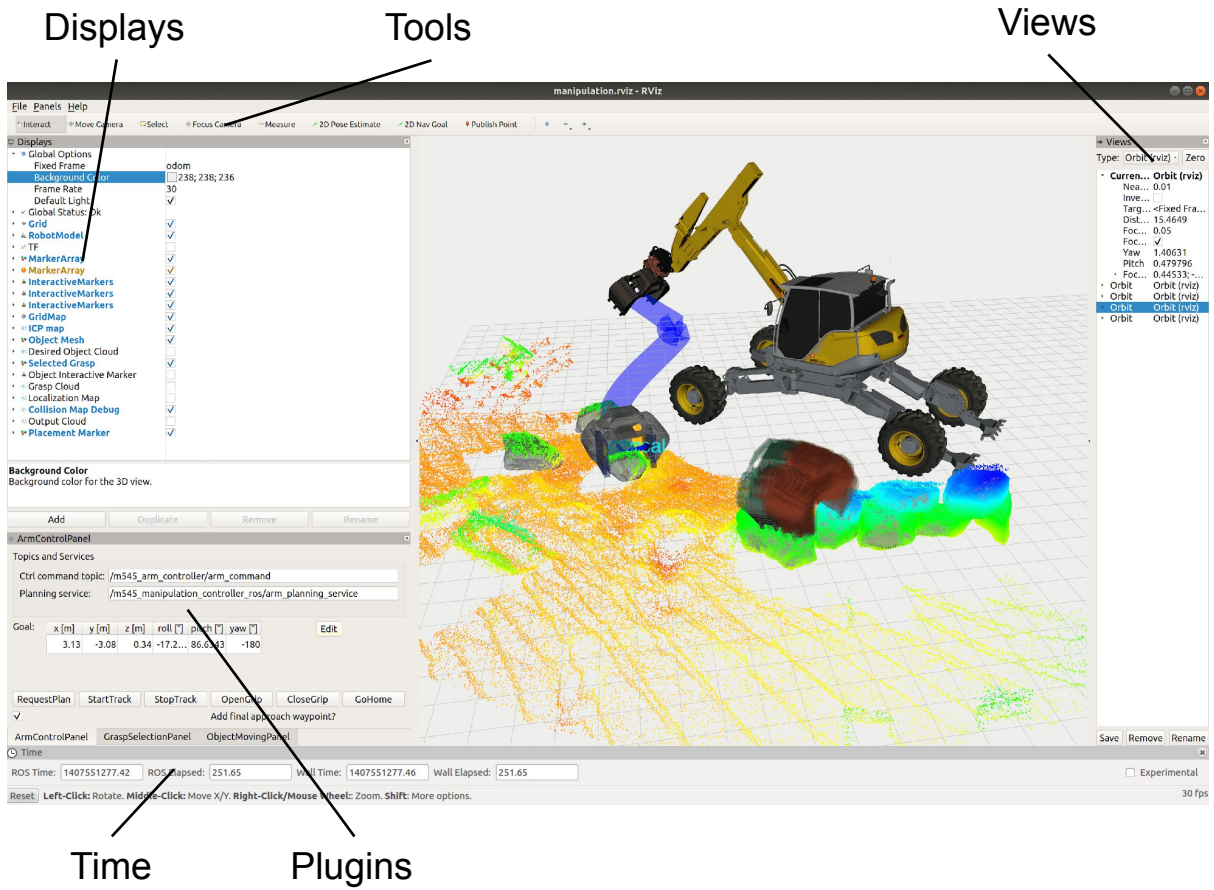
```
float64 x
float64 y
float64 z
float64 w
```

RViz

- 3D visualization tool for ROS
- Subscribes to topics and visualizes the message contents
- Different camera views (orthographic, top-down, etc.)
- Interactive tools to publish user information
- Save and load setup as RViz configuration
- Extensible with plugins

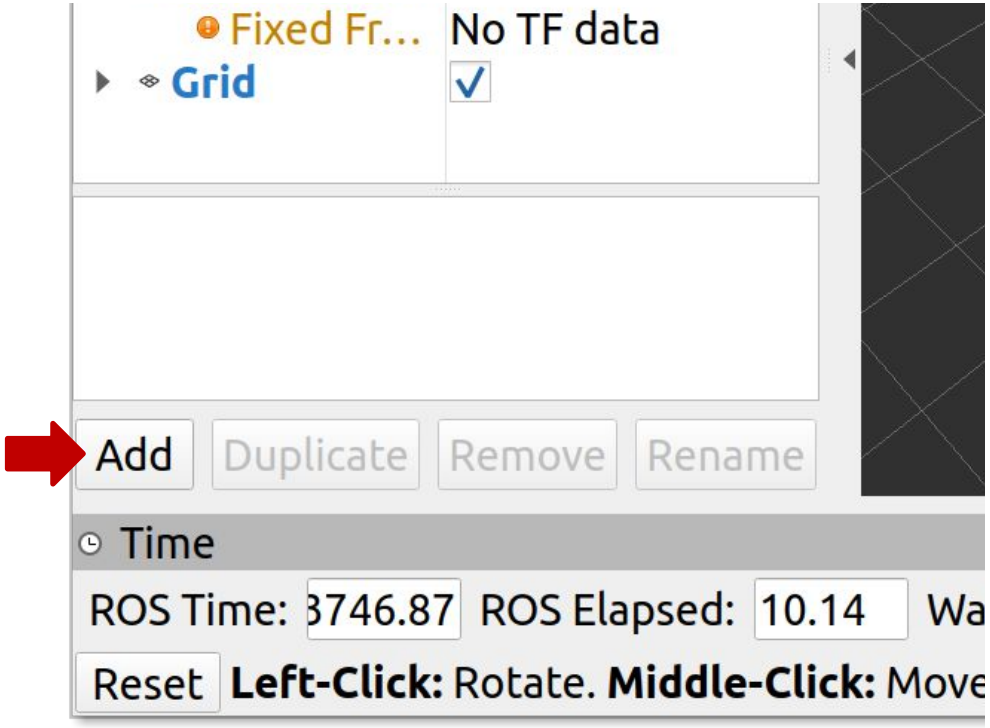
Run RViz with

```
> rviz
```



More info
wiki.ros.org/rviz

RViz
Display Plugins



Save configuration with `Ctrl + S`

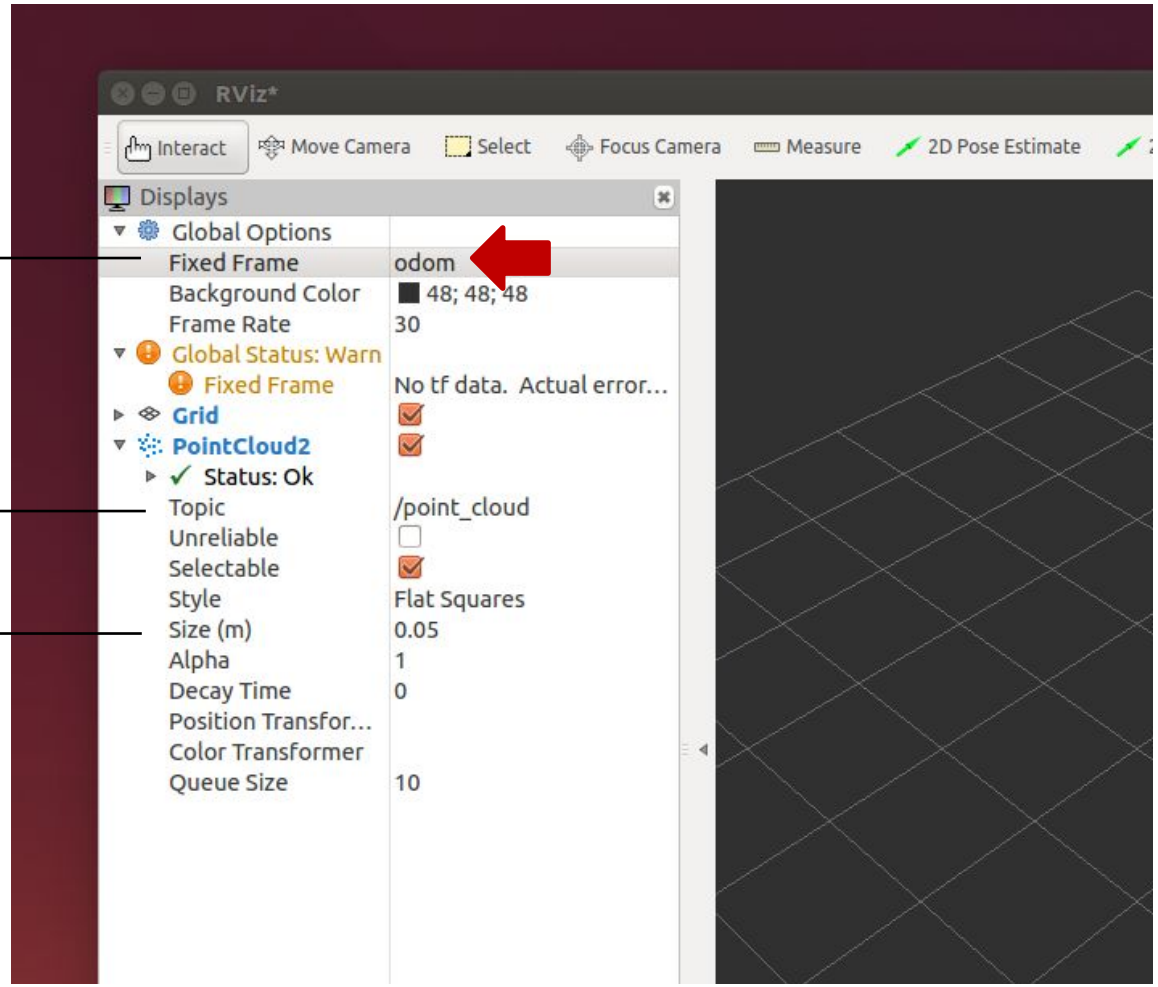
- Axes
- Camera
- DepthCloud
- Effort
- FluidPressure
- Grid
- GridCells
- Group
- Illuminance
- Image
- InteractiveMarkers
- LaserScan
- Map
- Marker
- MarkerArray
- Odometry
- Path
- PointCloud
- PointCloud2
- PointStamped
- Polygon
- Pose
- PoseArray
- Range
- RelativeHumidity
- RobotModel
- TF
- Temperature
- WrenchStamped

RViz
Visualizing Point Clouds Example

! Frame in which the data is displayed (has to exist!)

Choose the topic for the display

Change the display options (e.g. size)



ROS Packages

Code in ROS is organised in *packages*. All the packages installed by ROS are available in `/opt/ros/noetic/`. The custom ones you want to make can be placed in one or more *workspaces* in your home folder.

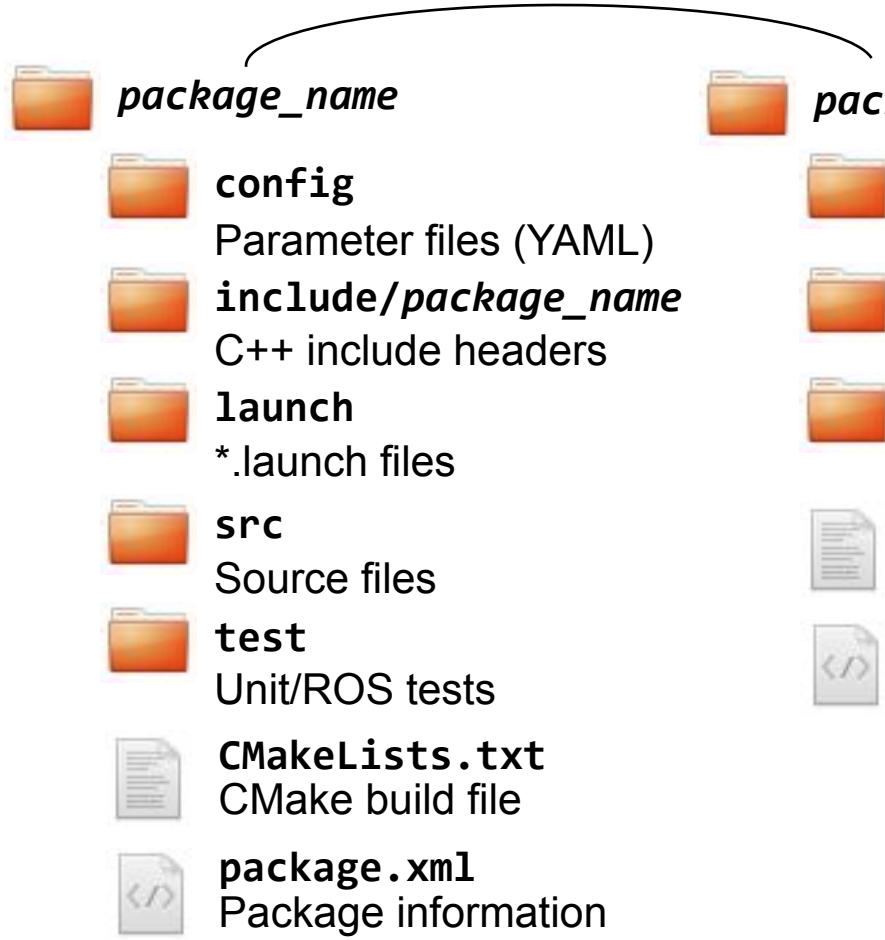
ROS Packages

- ROS software is organized into *packages*, which can contain source code, launch files, configuration files, message definitions, data, and documentation
- A package that builds up on/requires other packages (e.g. message definitions), declares these as *dependencies*

To create a new package, use

```
> catkin_create_pkg package_name
    {dependencies}
```

Separate message definition packages from other packages!



More info
wiki.ros.org/Packages

ROS Packages
package.xml

- The `package.xml` file defines the properties of the package
  - Package name
  - Version number
  - Authors
  - **Dependencies on other packages**
  - …

package.xml

```
<?xml version="1.0"?>
<package format="2">
   <name>ros_package_template</name>
   <version>0.1.0</version>
   <description>A ROS package that...</description>
   <maintainer email="tlankhorst@ethz.ch">Tom Lankhorst</maintainer>
   <license>BSD</license>
   <url type="website">https://github.com/leggedrobotics/ros_…</url>
   <author email="tlankhorst@ethz.ch">Tom Lankhorst</author>

   <buildtool_depend>catkin</buildtool_depend>

   <depend>roscpp</depend>
   <depend>std_msgs</depend>

   <build_depend>message_generation</build_depend>
</package>
```

More info
wiki.ros.org/catkin/package.xml

ROS Packages
CMakeLists.xml

The `CMakeLists.txt` is input to the CMake build system

1. Required CMake Version (`cmake_minimum_required`)
2. Package Name (`project()`)
3. Configure C++ standard and compile features
4. Find other CMake/Catkin packages needed for build (`find_package()`)
5. Message/Service/Action Generators (`add_message_files()`, `add_service_files()`, `add_action_files()`)
6. Invoke message/service/action generation (`generate_messages()`)
7. Specify package build info export (`catkin_package()`)
8. Libraries/Executables to build (`add_library()`/`add_executable()`/`target_link_libraries()`)
9. Tests to build (`catkin_add_gtest()`)
10. Install rules (`install()`)

CMakeLists.txt

```
cmake_minimum_required (VERSION 3.10.2 )
project (ros_package_template )

## Use C++14, or 11…
set (CMAKE_CXX_STANDARD 14 )
set (CMAKE_CXX_STANDARD_REQUIRED TRUE )

## Find catkin macros and libraries
find_package (catkin REQUIRED
        COMPONENTS
        roscpp
        sensor_msgs
        )
…
```

More info
wiki.ros.org/catkin/CMakeLists.txt

ROS Packages
CMakeLists.xml Example

```
cmake_minimum_required(VERSION 3.10.2)
project(smb_highlevel_controller)

set(CMAKE_CXX_STANDARD 11)
set(CMAKE_CXX_STANDARD_REQUIRED TRUE)

find_package(catkin REQUIRED
       COMPONENTS roscpp    sensor_msgs
       )

catkin_package(
       INCLUDE_DIRS include
       # LIBRARIES
       CATKIN_DEPENDS roscpp sensor_msgs
       # DEPENDS
)

include_directories(include ${catkin_INCLUDE_DIRS})

add_executable(${PROJECT_NAME}
src/${PROJECT_NAME}_node.cpp
src/SmbHighlevelController.cpp)

target_link_libraries(${PROJECT_NAME} ${catkin_LIBRARIES})
```

Use the same name as in the `package.xml`

Use C++11 by default (or 14)

List the packages that your package requires to build (have to be listed in `package.xml`)

Specify build export information
- `INCLUDE_DIRS`: Directories with header files
- `LIBRARIES`: Libraries created in this project
- `CATKIN_DEPENDS`: Packages dependent projects also need
- `DEPENDS`: System dependencies dependent projects also need (have to be listed in `package.xml`)

Specify locations of header files

Declare an executable, the node, with two src files

Specify libraries to link the executable against

ROS Launch

- *launch* is a tool for launching multiple nodes (as well as setting parameters)
- Are written in XML as **.launch* files
- If not yet running, launch automatically starts a roscore

Browse to the folder and start a launch file with

```
> roslaunch file_name.launch
```

Start a launch file from a package with

```
> roslaunch package_name file_name.launch
```

More info
http://wiki.ros.org/roslaunch

Example console output for
`roslaunch roscpp_tutorials talker_listener.launch`

```
student@ubuntu:~/catkin_ws$ roslaunch roscpp_tutorials talker_listener.launch
... logging to /home/student/.ros/log/794321aa-e950-11e6-95db-000c297bd368/rosl
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ubuntu:37592/

SUMMARY
========

PARAMETERS
 * /rosdistro: indigo
 * /rosversion: 1.11.20

NODES
  /
    listener (roscpp_tutorials/listener)
    talker (roscpp_tutorials/talker)

auto-starting new master
process[master]: started with pid [5772]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to 794321aa-e950-11e6-95db-000c297bd368
process[rosout-1]: started with pid [5785]
started core service [/rosout]
process[listener-2]: started with pid [5788]
process[talker-3]: started with pid [5795]
[ INFO] [1486044252.537801350]: hello world 0
[ INFO] [1486044252.638886504]: hello world 1
[ INFO] [1486044252.738279674]: hello world 2
[ INFO] [1486044252.838357245]: hello world 3
```

ROS Launch
File Structure

! Attention when copy & pasting code from the internet

talker_listener.launch

```
<launch>
    <node name="listener" pkg="roscpp_tutorials" type="listener" output="screen"/>
    <node name="talker" pkg="roscpp_tutorials" type="talker" output="screen"/>
</launch>
```

! Notice the syntax difference for self-closing tags: `<tag></tag>` and `<tag/>`

- **`launch`**: Root element of the launch file
- **`node`**: Each *<node>* tag specifies a node to be launched
- **`name`**: Name of the node (free to choose)
- **`pkg`**: Package containing the node
- **`type`**: Type of the node, there must be a corresponding executable with the same name
- **`output`**: Specifies where to output log messages (`screen`: console, `log`: log file)

More info
http://wiki.ros.org/roslaunch/XML
http://wiki.ros.org/roslaunch/Tutorials/Roslaunch%20tips%20for%20larger%20projects

ROS Launch
Arguments

- Create re-usable launch files with <arg> tag, which works like a parameter (default optional)

```
<arg name="arg_name" default="default_value"/>
```

- Use arguments in launch file with

```
$(arg arg_name)
```

- When launching, arguments can be set with

```
> roslaunch launch_file.launch arg_name:=value
```

range_world.launch (simplified)

```
<?xml version="1.0"?>
<launch>
  <arg name="use_sim_time" default="true"/>
  <arg name="world" default="gazebo_ros_range"/>
  <arg name="debug" default="false"/>
  <arg name="physics" default="ode"/>

  <group if="$(arg use_sim_time)">
    <param name="/use_sim_time" value="true" />
  </group>

  <include file="$(find gazebo_ros)
                                 /launch/empty_world.launch">
    <arg name="world_name" value="$(find gazebo_plugins)/
                      test/test_worlds/$(arg world).world"/>
    <arg name="debug" value="$(arg debug)"/>
    <arg name="physics" value="$(arg physics)"/>
  </include>
</launch>
```

More info
http://wiki.ros.org/roslaunch/XML/arg

ROS Launch
Including Other Launch Files

- Include other launch files with <include> tag to organize large projects

```
<include file="package_name"/>
```

- Find the system path to other packages with

```
$(find package_name)
```

- Pass arguments to the included file

```
<arg name="arg_name" value="value"/>
```

range_world.launch (simplified)

```
<?xml version="1.0"?>
<launch>
  <arg name="use_sim_time" default="true"/>
  <arg name="world" default="gazebo_ros_range"/>
  <arg name="debug" default="false"/>
  <arg name="physics" default="ode"/>

  <group if="$(arg use_sim_time)">
    <param name="/use_sim_time" value="true" />
  </group>

  <include file="$(find gazebo_ros)
                                 /launch/empty_world.launch">
    <arg name="world_name" value="$(find gazebo_plugins)/
                      test/test_worlds/$(arg world).world"/>
    <arg name="debug" value="$(arg debug)"/>
    <arg name="physics" value="$(arg physics)"/>
  </include>
</launch>
```

More info
http://wiki.ros.org/roslaunch/XML/include

ROS Parameter Server

- Nodes use the *parameter server* to store and retrieve parameters at runtime
- Best used for static data such as configuration parameters
- Parameters can be defined in launch files or separate *YAML* files

List all parameters with

```
> rosparam list
```

Get the value of a parameter with

```
> rosparam get parameter_name
```

Set the value of a parameter with

```
> rosparam set parameter_name value
```

config.yaml

```
camera:
  left:
    name: left_camera
    exposure: 1
  right:
    name: right_camera
    exposure: 1.1
```

package.launch

```
<launch>
  <node name="name" pkg="package" type="node_type">
      <rosparam command="load"
                file="$(find package)/config/config.yaml" />
  </node>
</launch>
```

More info
wiki.ros.org/rosparam

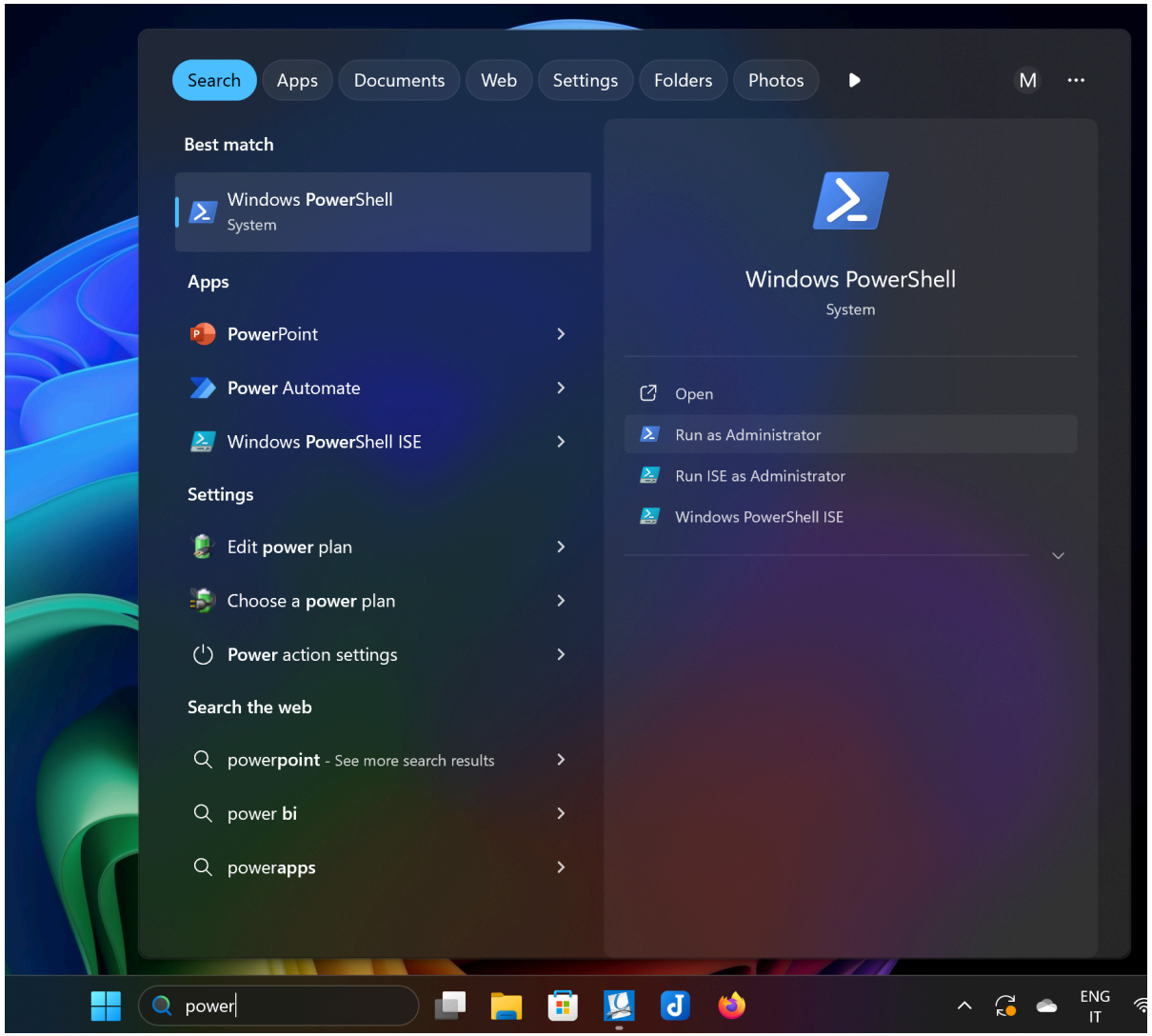
Interactive Session: Let's play with ROS

Using Ubuntu on a Windows machine with WSL

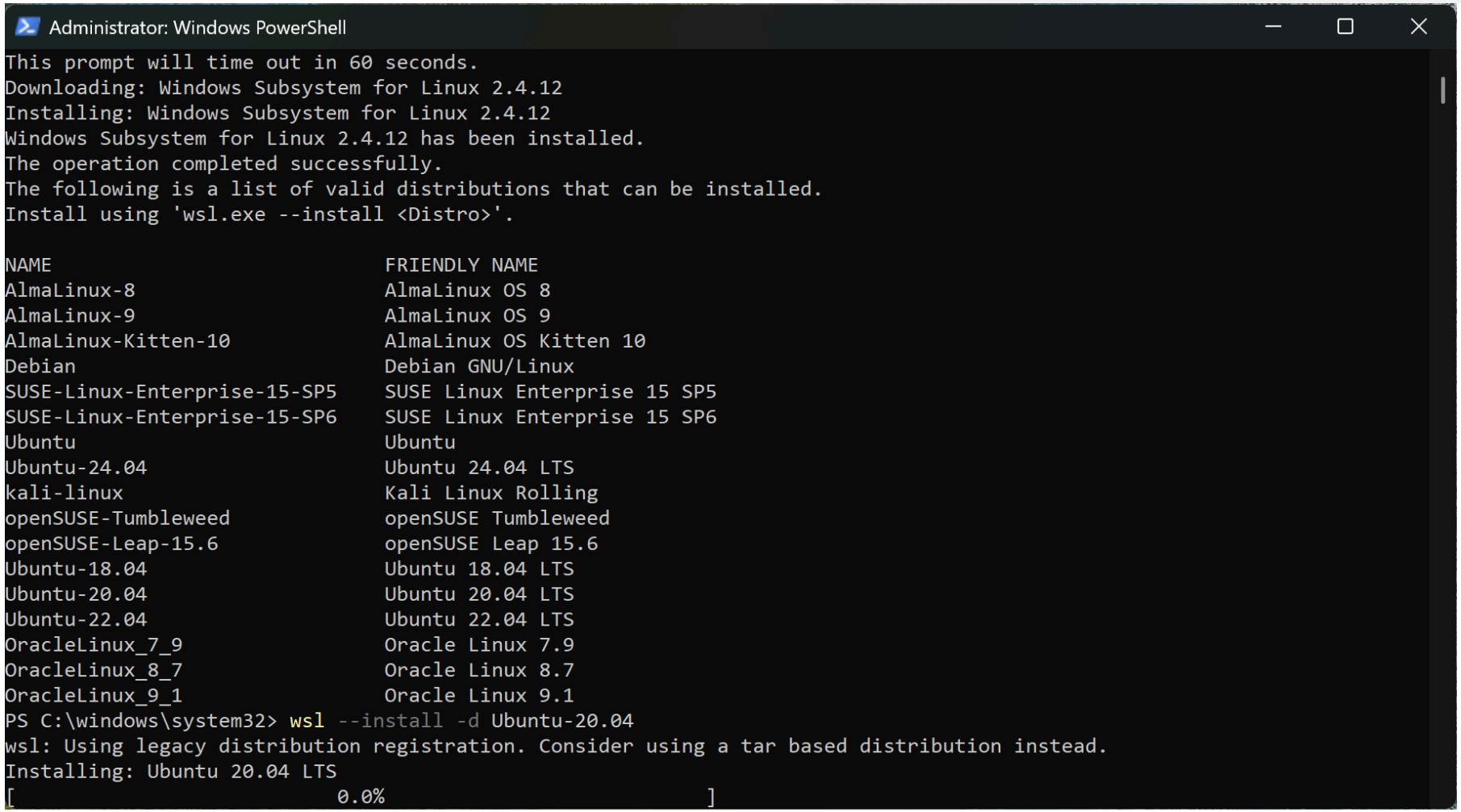
- For this course we will use Ubuntu 20.04 (Focal Fossa).
- If you have a Windows machine and you don't want to install Ubuntu natively on your computer, you can run Ubuntu from within Windows using Windows Subsystem for Linux (WSL)
- For those who want to try out Ubuntu on their machine directly, making a bootable USB flash drive is the preferred method. More info on this here: https://ubuntu.com/tutorials/install-ubuntu-desktop#1-overview
- Mac users can try UTM to install Ubuntu on their system: https://dev.to/andrewbaisden/how-to-install-ubuntu-linux-on-apple-silicon-macbooks-1nia

Use Ubuntu on Windows with WSL

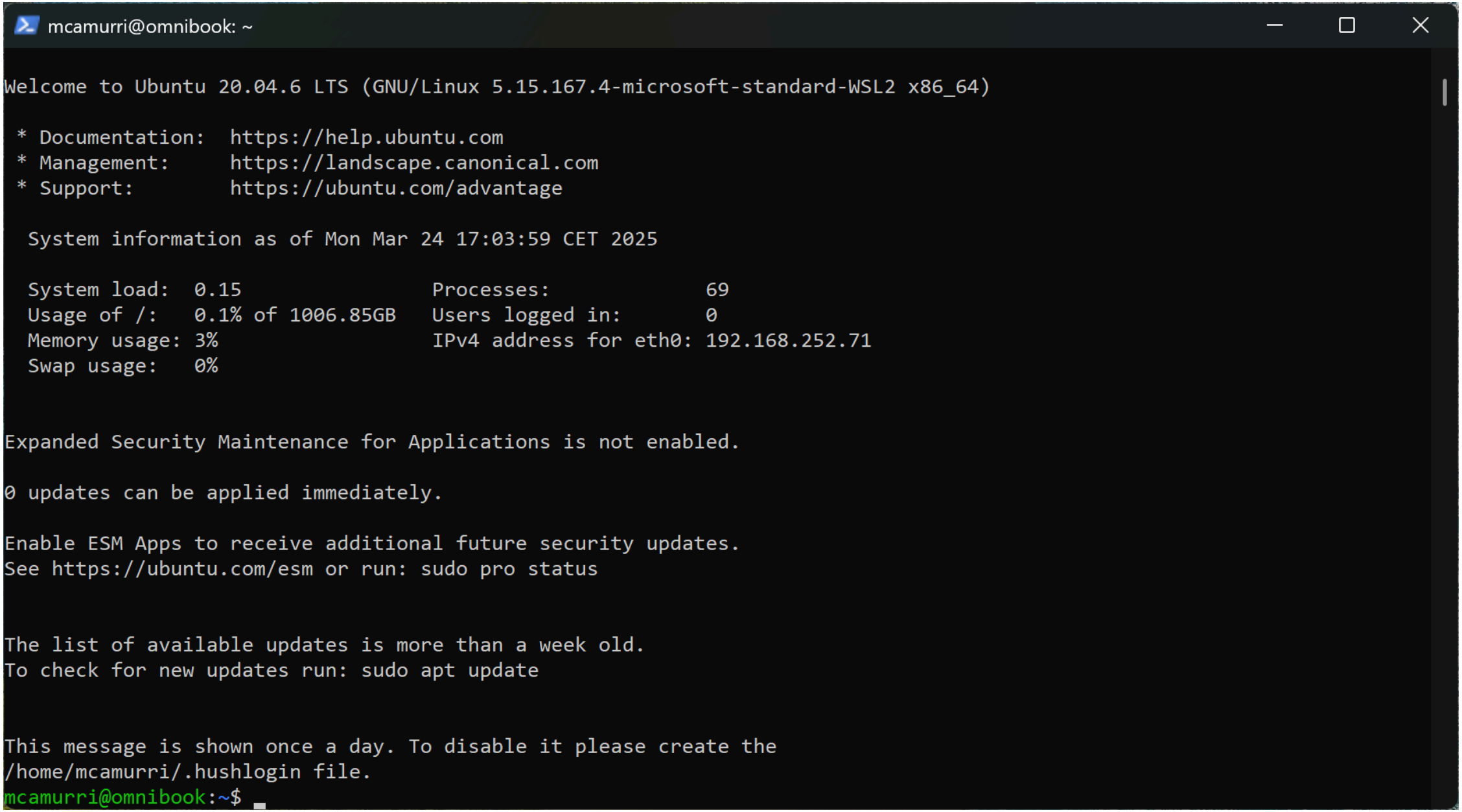
1. Type `PowerShell` on the search bar
2. Start the Windows PowerShell as administrator by selecting "Run as Administrator" on the right panel. Your password will be required.



3. On the powersheell, run the command `wsl --install`
4. After it is installed, run the command `wsl --install -d Ubuntu-20.04`



5. During installation type the username and password for your system. The username is normally all lowercase, one word, e.g. `mcamurri`. See next slide.
6. Once the system is installed, the prompt should change to the linux one, which is mostly green and looks like this: `username@hostname:~$` where `username` is the user name you chose previously and `hostname` is the name of your computer, chosen automatically for you by WSL.


mcamurri@omnibook: ~
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.15.167.4-microsoft-standard-WSL2 x86_64)
* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/advantage
System information as of Mon Mar 24 17:03:59 CET 2025
System load: 0.15
Usage of /: 0.1% of 1006.85GB
Memory usage: 3%
Swap usage: 0%
Processes: 69
Users logged in: 0
IPv4 address for eth0: 192.168.252.71
Expanded Security Maintenance for Applications is not enabled.
0 updates can be applied immediately.
Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status
The list of available updates is more than a week old.
To check for new updates run: sudo apt update
This message is shown once a day. To disable it please create the
/home/mcamurri/.hushlogin file.
mcamurri@omnibook:~$

Installing ROS Noetic - Impatient Edition

The following instructions are also available here.
Just type the following commands one by one and press Enter.

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

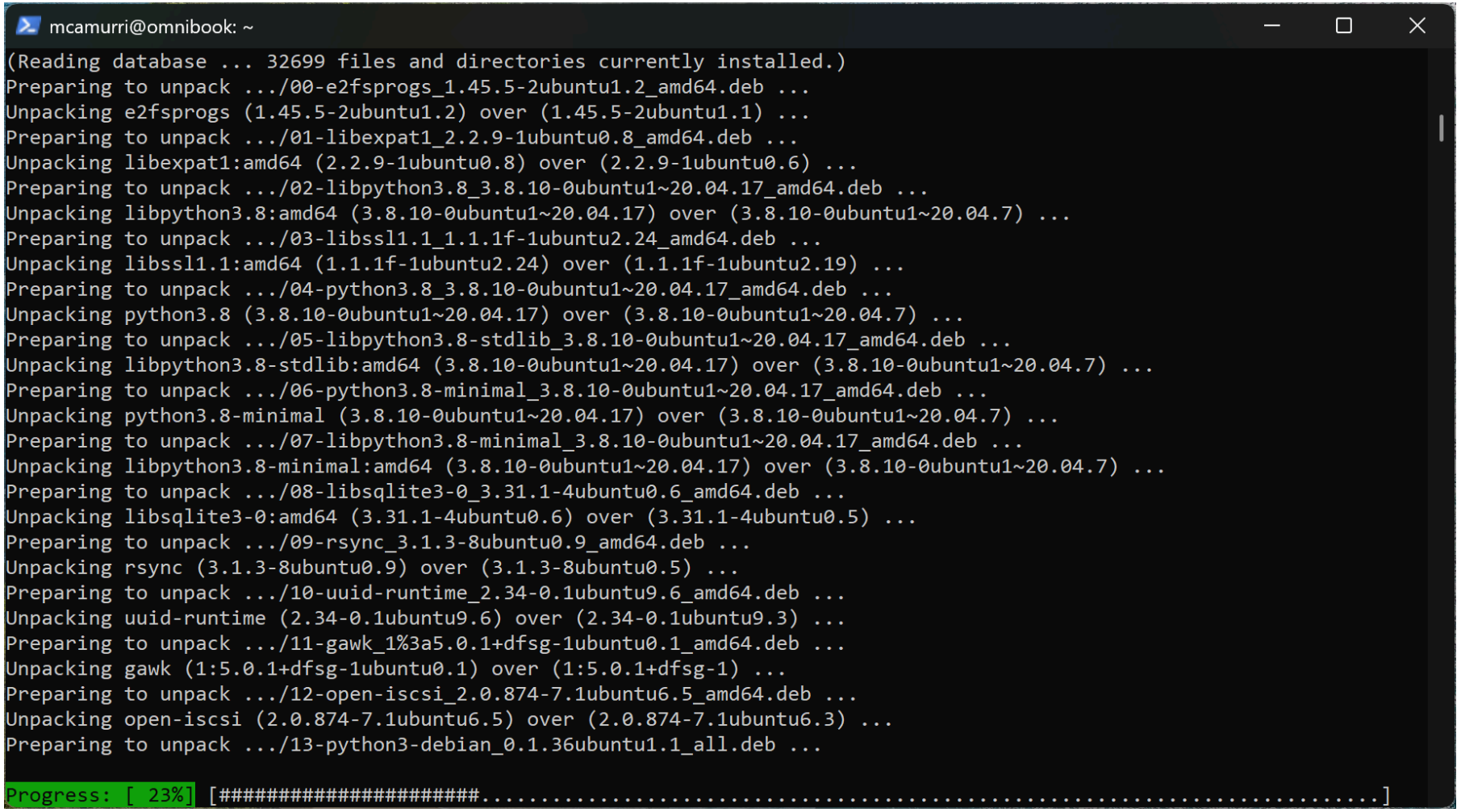
```
sudo apt install curl
```

```
curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -
```

```
sudo apt update
```

```
sudo apt install ros-noetic-desktop-full
```

Wait for it …


mcamurri@omnibook: ~
(Reading database ... 32699 files and directories currently installed.)
Preparing to unpack .../00-e2fsprogs_1.45.5-2ubuntu1.2_amd64.deb ...
Unpacking e2fsprogs (1.45.5-2ubuntu1.2) over (1.45.5-2ubuntu1.1) ...
Preparing to unpack .../01-libexpat1_2.2.9-1ubuntu0.8_amd64.deb ...
Unpacking libexpat1:amd64 (2.2.9-1ubuntu0.8) over (2.2.9-1ubuntu0.6) ...
Preparing to unpack .../02-libpython3.8_3.8.10-0ubuntu1~20.04.17_amd64.deb ...
Unpacking libpython3.8:amd64 (3.8.10-0ubuntu1~20.04.17) over (3.8.10-0ubuntu1~20.04.7) ...
Preparing to unpack .../03-libssl1.1_1.1.1f-1ubuntu2.24_amd64.deb ...
Unpacking libssl1.1:amd64 (1.1.1f-1ubuntu2.24) over (1.1.1f-1ubuntu2.19) ...
Preparing to unpack .../04-python3.8_3.8.10-0ubuntu1~20.04.17_amd64.deb ...
Unpacking python3.8 (3.8.10-0ubuntu1~20.04.17) over (3.8.10-0ubuntu1~20.04.7) ...
Preparing to unpack .../05-libpython3.8-stdlib_3.8.10-0ubuntu1~20.04.17_amd64.deb ...
Unpacking libpython3.8-stdlib:amd64 (3.8.10-0ubuntu1~20.04.17) over (3.8.10-0ubuntu1~20.04.7) ...
Preparing to unpack .../06-python3.8-minimal_3.8.10-0ubuntu1~20.04.17_amd64.deb ...
Unpacking python3.8-minimal (3.8.10-0ubuntu1~20.04.17) over (3.8.10-0ubuntu1~20.04.7) ...
Preparing to unpack .../07-libpython3.8-minimal_3.8.10-0ubuntu1~20.04.17_amd64.deb ...
Unpacking libpython3.8-minimal:amd64 (3.8.10-0ubuntu1~20.04.17) over (3.8.10-0ubuntu1~20.04.7) ...
Preparing to unpack .../08-libsqlite3-0_3.31.1-4ubuntu0.6_amd64.deb ...
Unpacking libsqlite3-0:amd64 (3.31.1-4ubuntu0.6) over (3.31.1-4ubuntu0.5) ...
Preparing to unpack .../09-rsync_3.1.3-8ubuntu0.9_amd64.deb ...
Unpacking rsync (3.1.3-8ubuntu0.9) over (3.1.3-8ubuntu0.5) ...
Preparing to unpack .../10-uuid-runtime_2.34-0.1ubuntu9.6_amd64.deb ...
Unpacking uuid-runtime (2.34-0.1ubuntu9.6) over (2.34-0.1ubuntu9.3) ...
Preparing to unpack .../11-gawk_1%3a5.0.1+dfsg-1ubuntu0.1_amd64.deb ...
Unpacking gawk (1:5.0.1+dfsg-1ubuntu0.1) over (1:5.0.1+dfsg-1) ...
Preparing to unpack .../12-open-iscsi_2.0.874-7.1ubuntu6.5_amd64.deb ...
Unpacking open-iscsi (2.0.874-7.1ubuntu6.5) over (2.0.874-7.1ubuntu6.3) ...
Preparing to unpack .../13-python3-debian_0.1.36ubuntu1.1_all.deb ...
Progress: [23%] [#######################.....]

Example

Console Tab Nr. 1 – Starting a *roscore*

Start a roscore with

```
> roscore
```

```
student@ubuntu:~/catkin_ws$ roscore
... logging to /home/student/.ros/log/6c1852aa-e961-11e6-8543-000c297bd368/ros
launch-ubuntu-6696.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ubuntu:34089/
ros_comm version 1.11.20


SUMMARY
========

PARAMETERS
 * /rosdistro: indigo
 * /rosversion: 1.11.20

NODES

auto-starting new master
process[master]: started with pid [6708]
ROS_MASTER_URI=http://ubuntu:11311/

setting /run_id to 6c1852aa-e961-11e6-8543-000c297bd368
process[rosout-1]: started with pid [6721]
started core service [/rosout]
```

Example

Console Tab Nr. 2 – Starting a *talker* node

Run a talker demo node with

```
> rosrun roscpp_tutorials talker
```

```
student@ubuntu:~/catkin_ws$ rosrun roscpp_tutorials talker
[ INFO] [1486051708.424661519]: hello world 0
[ INFO] [1486051708.525227845]: hello world 1
[ INFO] [1486051708.624747612]: hello world 2
[ INFO] [1486051708.724826782]: hello world 3
[ INFO] [1486051708.825928577]: hello world 4
[ INFO] [1486051708.925379775]: hello world 5
[ INFO] [1486051709.024971132]: hello world 6
[ INFO] [1486051709.125450960]: hello world 7
[ INFO] [1486051709.225272747]: hello world 8
[ INFO] [1486051709.325389210]: hello world 9
```

Example

Console Tab Nr. 3 – Analyze *talker* node

See the list of active nodes

```
> rosnode list
```

```
student@ubuntu:~/catkin_ws$ rosnode list
/rosout
/talker
```

Show information about the *talker* node

```
> rosnode info /talker
```

```
student@ubuntu:~/catkin_ws$ rosnode info /talker
--------------------------------------------------------------------------------
Node [/talker]
Publications:
 * /chatter [std_msgs/String]
 * /rosout [rosgraph_msgs/Log]

Subscriptions: None

Services:
 * /talker/get_loggers
 * /talker/set_logger_level
```

See information about the *chatter* topic

```
> rostopic info /chatter
```

```
student@ubuntu:~/catkin_ws$ rostopic info /chatter
Type: std_msgs/String

Publishers:
 * /talker (http://ubuntu:39173/)

Subscribers: None
```

Example

Console Tab Nr. 3 – Analyze *chatter* topic

Check the type of the *chatter* topic

```
> rostopic type /chatter
```

```
student@ubuntu:~/catkin_ws$ rostopic type /chatter
std_msgs/String
```

Show the message contents of the topic

```
> rostopic echo /chatter
```

```
student@ubuntu:~/catkin_ws$ rostopic echo /chatter
data: hello world 11874
---
data: hello world 11875
---
data: hello world 11876
```

Analyze the frequency

```
> rostopic hz /chatter
```

```
student@ubuntu:~/catkin_ws$ rostopic hz /chatter
subscribed to [/chatter]
average rate: 9.991
        min: 0.099s max: 0.101s std dev: 0.00076s window: 10
average rate: 9.996
        min: 0.099s max: 0.101s std dev: 0.00069s window: 20
```

Example

Console Tab Nr. 4 – Starting a *listener* node

Run a listener demo node with

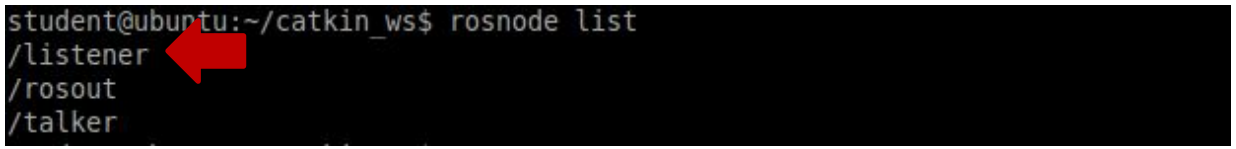
```
> rosrun roscpp_tutorials listener
```

```
student@ubuntu:~/catkin_ws$ rosrun roscpp_tutorials listener
[ INFO] [1486053802.204104598]: I heard: [hello world 19548]
[ INFO] [1486053802.304538827]: I heard: [hello world 19549]
[ INFO] [1486053802.403853395]: I heard: [hello world 19550]
[ INFO] [1486053802.504438133]: I heard: [hello world 19551]
[ INFO] [1486053802.604297608]: I heard: [hello world 19552]
```

Example
Console Tab Nr. 3 – Analyze

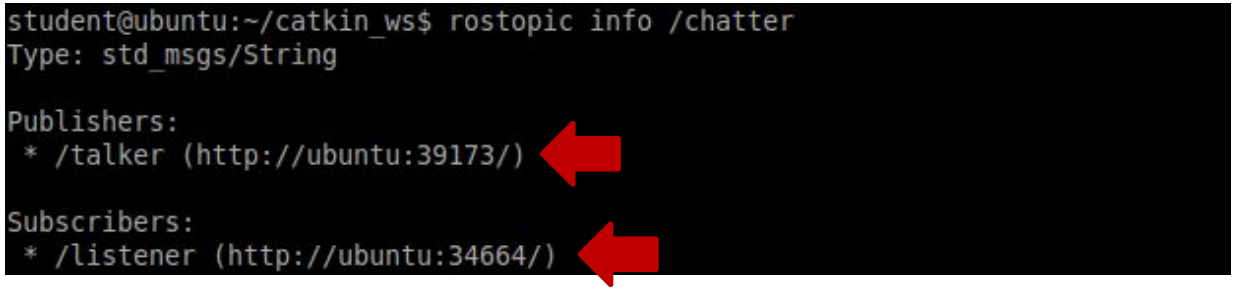
See the new *listener* node with

```
> rosnode list
```



Show the connection of the nodes over the chatter topic with

```
> rostopic info /chatter
```



Example
Console Tab Nr. 3 – Publish Message from Console

Close the *talker* node in console nr. 2 with `Ctrl + C`

Publish your own message with

```
> rostopic pub /chatter std_msgs/String
  "data: 'ETH Zurich ROS Course'"
```

```
student@ubuntu:~/catkin_ws$ rostopic pub /chatter std_msgs/String "data: 'ETH
Zurich ROS Course'"
publishing and latching message. Press ctrl-C to terminate
```

Check the output of the *listener* in console nr. 4

```
[ INFO] [1486054667.604322265]: I heard: [hello world 28202]
[ INFO] [1486054667.704264199]: I heard: [hello world 28203]
[ INFO] [1486054667.804389058]: I heard: [hello world 28204]
[ INFO] [1486054707.646404558]: I heard: [ETH Zurich ROS Course]
```

ROS Workspace Environment

- Defines context for the current workspace
- Default workspace loaded with

```
> source /opt/ros/noetic/setup.bash
```

Overlay your *catkin workspace* with

```
> cd ~/catkin_ws
> source devel/setup.bash
```

Check your workspace with

```
> echo $ROS_PACKAGE_PATH
```

This is already setup in the provided installation.

See setup with

```
> cat ~/.bashrc
```

More info

http://wiki.ros.org/kinetic/Installation/Ubuntu
http://wiki.ros.org/catkin/workspaces

catkin Build System

- *catkin* is the ROS build system to generate executables, libraries, and interfaces
- We suggest to use the *Catkin Command Line Tools*

→ Use **`catkin build`** instead of `catkin_make`

Navigate to your catkin workspace with

```
> cd ~/catkin_ws
```

Build a package with

```
> catkin build package_name
```

! Whenever you build a **new** package, update your environment

```
> source devel/setup.bash
```

The catkin command line tools are pre-installed in the provided installation.

More info
http://wiki.ros.org/catkin/Tutorials
https://catkin-tools.readthedocs.io/

catkin Build System

The catkin workspace contains the following spaces

Work here

The *source space* contains the source code. This is where you can clone, create, and edit source code for the packages you want to build.

Don't touch

The *build space* is where CMake is invoked to build the packages in the source space. Cache information and other intermediate files are kept here.

Don't touch

The *development (devel) space* is where built targets are placed (prior to being installed).

If necessary, clean the entire build and devel space with

```
> catkin clean
```

More info
http://wiki.ros.org/catkin/workspaces

catkin Build System

The catkin workspace setup can be checked with

```
> catkin config
```

For example, to set the *CMake build type* to Release (or Debug etc.), use

```
> catkin build --cmake-args
      -DCMAKE_BUILD_TYPE=Release
```

```
student@ubuntu:~/catkin_ws$ catkin config
--------------------------------------------------------------------------------
Profile:                     default
Extending:             [env] /opt/ros/indigo:/home/student/catkin_ws/devel
Workspace:                   /home/student/catkin_ws
--------------------------------------------------------------------------------
Source Space:       [exists] /home/student/catkin_ws/src
Log Space:          [exists] /home/student/catkin_ws/logs
Build Space:        [exists] /home/student/catkin_ws/build
Devel Space:        [exists] /home/student/catkin_ws/devel
Install Space:      [unused] /home/student/catkin_ws/install
DESTDIR:            [unused] None
--------------------------------------------------------------------------------
Devel Space Layout:          linked
Install Space Layout:        None
--------------------------------------------------------------------------------
Additional CMake Args:       -GEclipse CDT4 - Unix Makefiles -DCMAKE_CXX_COMP
ILER_ARG1=-std=c++11 -DCMAKE_BUILD_TYPE=Release
Additional Make Args:        None
Additional catkin Make Args: None
Internal Make Job Server:    True
Cache Job Environments:      False
--------------------------------------------------------------------------------
Whitelisted Packages:        None
Blacklisted Packages:        None
--------------------------------------------------------------------------------
Workspace configuration appears valid.
--------------------------------------------------------------------------------
```

Already setup in the provided installation.

More info
http://catkin-tools.readthedocs.io/en/latest/verbs/catkin_config.html
http://catkin-tools.readthedocs.io/en/latest/cheat_sheet.html

Example

Open a terminal and browse to your `git` folder

```
> cd ~/git
```

Clone the Git repository with

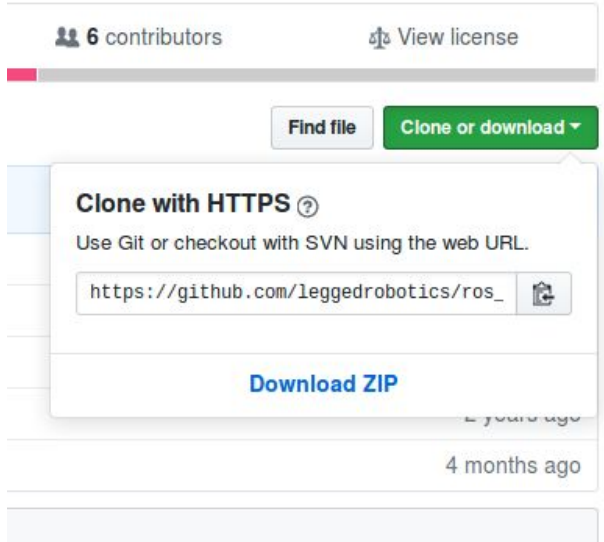
```
> git clone https://github.com/leggedrobotics/
ros_best_practices.git
```

Symlink the new package to your catkin workspace

```
> ln -s ~/git/ros_best_practices/
 ~/catkin_ws/src/
```

Note: You could also directly clone to your catkin workspace, but using a common `git` folder is convenient if you have multiple catkin workspaces.

https://github.com/leggedrobotics/ros_best_practices



Example

Go to your catkin workspace

```
> cd ~/catkin_ws
```

Build the package with

```
> catkin build ros_package_template
```

Re-source your workspace setup

```
> source devel/setup.bash
```

Launch the node with

```
> roslaunch ros_package_template
  ros_package_template.launch
```

```
NOTE: Forcing CMake to run for each package.
--------------------------------------------------------------------------------
[build] Found '1' packages in 0.0 seconds.
[build] Updating package table.
Starting  >>> catkin_tools_prebuild
Finished  <<< catkin_tools_prebuild                [ 1.0 seconds ]
Starting  >>> ros_package_template
Finished  <<< ros_package_template                 [ 4.1 seconds ]
[build] Summary: All 2 packages succeeded!
[build]   Ignored:   None.
[build]   Warnings:  None.
[build]   Abandoned: None.
[build]   Failed:    None.
[build] Runtime: 5.2 seconds total.
[build] Note: Workspace packages have changed, please re-source setup files to u
se them.
student@ubuntu:~/catkin_ws$
```

```
 * /rosdistro: indigo
 * /rosversion: 1.11.20

NODES
  /
    ros_package_template (ros_package_template/ros_package_template)

auto-starting new master
process[master]: started with pid [27185]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to e43f937a-ed52-11e6-9789-000c297bd368
process[rosout-1]: started with pid [27198]
started core service [/rosout]
process[ros_package_template-2]: started with pid [27201]
[ INFO] [1486485095.843512614]: Successfully launched node.
```