



## 02. Linux and ROS principles

### Lecture

#### Linux principles



- Only OS supported by ROS
- Security
- Efficiency
- Open-source
- Community support
- User freedom
- Distributions: **Ubuntu**, Linux Mint, Debian, etc.
- Terminal usage more dominant

#### Suggestion

Install **Terminator** terminal emulator:

```
sudo apt update  
sudo apt install terminator
```

## Linux commands

See some basic commands below:

- Run as administrator with `sudo`
- Manual of command `man` , e.g. `man cp`
- Package management `apt` , e.g. `apt update` , `apt install`
- Navigation `cd`
- List directory contents `ls`
- Copy file `cp`
- Move file `mv`
- Remove file `rm`
- Make directory `mkdir`
- Remove directory `rmdir`
- Make a file executable `chmod +x <filename>`
- Safe restart: Ctrl + Alt + PrtScr + REISUB
- If not sure, just google the command

## ROS principles

### ROS file system



### ROS package principle

Enough functionality to be useful, but not too much that the package is heavyweight and difficult to use from other software.

## ROS package

- Main unit to organize software in ROS
- Buildable and redistributable unit of ROS code
- Consists of:
  - Manifest (package.xml): information about package
    - name
    - version
    - description
    - dependencies
    - etc.
  - CMakeLists.txt: *input for the CMake build system*
  - Anything else
- `roslaunch turtlesim turtlesim_node`

## **ROS node**

- Executable part of ROS:
  - python scripts
  - compiled C++ code
- A process that performs computation
- Inter-node communication:
  - ROS topics (streams)
  - ROS parameter server
  - Remote Procedure Calls (RPC)
  - ROS services
  - ROS actions
- Meant to operate at a fine-grained scale
- Typically, a robot control system consists of many nodes, like:
  - Trajectory planning
  - Localization
  - Read sensory data
  - Process sensory data
  - Motor control
  - User interface
  - etc.

## **ROS build system---Catkin**

- System for building software packages in ROS



## ROS workspace

### Catkin workspace

A folder where catkin packages are modified, built, and installed.



- Source space:
  - Source code of catkin packages
  - Space where you can extract/checkout/clone source code for the packages you want to build
- Build space
  - CMake is invoked here to build the catkin packages

- CMake and catkin keep intermediate files here
- Devel space:
  - Built target are placed here prior to being installed

## Environmental setup file

- setup.bash
- generated during init process of a new workspace
- extends shell environment
- ROS can find any resources that have been installed or built to that location

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

## ROS master

```
roscore
```

- Registers:
  - Nodes
  - Topics
  - Services
  - Parameters
- One per system
- `roslaunch` launches ROS master automatically

## Gyakorlat



### Warning!

At the end of the lesson, everybody must upload their **sources** to **Moodle** as a zip archive!

## 1: Turtlesim

1. Launch ROS master, `turtlesim_node` and `turtle_teleop_key` node by typing the following commands to separate terminal windows:

### Tip

In **Terminator**, `Ctrl-Shift-O`, `Ctrl-Shift-E` divides the terminal window, `Ctrl-Shift-W` closes the current window.

```
roscore
roslaunch turtlesim turtlesim_node
roslaunch turtlesim turtle_teleop_key
```

### To stop running

`Ctrl-C`

2. Display the running nodes and topics using this command, in a separate terminal:

```
roslaunch rqt_graph rqt_graph
```

3. Try the following commands to gain more information about the currently running system:

```
roswtf
rospack list
rospack find turtlesim
roslaunch turtlesim turtlesim_node
roslaunch turtlesim turtle_teleop_key
rostopic list
rostopic info /turtle1/cmd_vel
rostopic echo /turtle1/cmd_vel
```

4. Type (or copy) the following command to the terminal:



```
rostopic pub /turtle1/cmd_vel geometry_msgs/Twist -r 1 -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'
```

## 2: Catkin workspace

1. Install the catkin build tools package:

```
sudo apt update  
sudo apt-get install python3-catkin-tools python3-osrf-pycommon
```

2. Copy the following file to the end of file `~/.bashrc` :

```
source /opt/ros/noetic/setup.bash # replace noetic by whatever your ROS distribution  
is
```

3. Create the workspace:

```
source /opt/ros/noetic/setup.bash  
mkdir -p ~/catkin_ws/src  
cd ~/catkin_ws  
catkin init
```

## 3: Create a new ROS package

1. Create a new ROS package named `ros_course` , depends packages `std_msgs` ,  
`rospy` and `roscpp` :

```
cd ~/catkin_ws/src  
catkin create pkg ros_course --catkin-deps std_msgs rospy roscpp
```



### Syntax

```
catkin create pkg <PKG_NAME> --catkin-deps <DEP_1> <DEP_2>
```

2. Open the file `package.xml` and fill the following tags:

```
<description>The beginner_tutorials package</description>

<maintainer email="you@yourdomain.tld">Your Name</maintainer>
```

### 3. Build the workspace:

```
cd ~/catkin_ws
catkin build
```

#### Danger

The commands `catkin build` and `catkin make` are not meant to be used within the same workspace!

### 4. Append the following line to the file `~/bashrc` :

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

## 4: Implement a Publisher in Python

### 1. Create folder named `scripts` in the `ros_course` package:

```
cd ~/catkin_ws/src/ros_course
mkdir scripts
cd scripts
```

### 2. Navigate to the `scripts` folder and create the file `talker.py` , fill it with the following content:

```
import rospy
from std_msgs.msg import String

def talker():
    rospy.init_node('talker', anonymous=True)
    pub = rospy.Publisher('chatter', String, queue_size=10)

    rate = rospy.Rate(10) # 10hz

    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
```

```

    print(hello_str)
    pub.publish(hello_str)
    rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass

```

3. Open `CMakeLists.txt` and find the commented out line starting with `catkin_install_python` (it is near line 167). Uncomment and edit as the following:

```

catkin_install_python(PROGRAMS scripts/talker.py
    DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
)

```

4. Build the node:

```

cd ~/catkin_ws
catkin build

```

5. Start ROS master and run the node. In separate terminal windows:

```
roscore
```

```
roslaunch ros_course talker.py
```

6. Check the output of the node, in a separate terminal:

```
rostopic echo chatter
```

## 5: Implement a Subscriber in Python

1. Navigate to the `scripts` folder and create the file `listener.py`:

```

import rospy
from std_msgs.msg import String

def callback(data):

```

```

print(rospy.get_caller_id() + "I heard " + data.data)

def listener():

    # In ROS, nodes are uniquely named. If two nodes with the same
    # name are launched, the previous one is kicked off. The
    # anonymous=True flag means that rospy will choose a unique
    # name for our 'listener' node so that multiple listeners can
    # run simultaneously.
    rospy.init_node('listener', anonymous=True)

    rospy.Subscriber("chatter", String, callback)

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()

if __name__ == '__main__':
    listener()

```

## 2. Modify CMakeLists.txt :

```

catkin_install_python(PROGRAMS scripts/talker.py scripts/listener.py
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
)

```

## 3. Build the workspace:

```

cd ~/catkin_ws
catkin build

```

## 4. Start ROS master and run the 2 nodes. In separate terminal windows:

```
roscore
```

```
roslaunch ros_course talker.py
```

```
roslaunch ros_course listener.py
```

## 5. Check the nodes and topic of the system:

```
roslaunch rqt_graph rqt_graph
```

### **Warning!**

At the end of the lesson, everybody must upload their **sources** to **Moodle** as a zip archive!

## Links

- [ROS Tutorials](#)
- [Curiosity rover simulation](#)