# 02. Linux, ROS alapismeretek

# Elmélet

# Linux principles



- (Was) the only OS supported by ROS
- Security
- Efficieny
- 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
- Create file touch
- Copy file cp
- Move file mv
- Remove file rm
- Make directory mkdir
- Remove directory rmdir
- Make a file executable chmod +x <filename>
- Safe restart: Crtl + Alt + PrtScr + REISUB
- If not sure, just google the command

### $ROS 1 \rightarrow ROS 2$

- ROS 2 was rewritten from scratch
- More modular architecture
- Improved support for real-time systems
- Support for multiple communication protocols
- Better interoperability with other robotic systems
- Focus on standardization and industry collaboration
- No ROS Master
- No Devel space
- rclpy, rclcpp
- More structured code (Node class)

- Different build system
- Platforms: Windows, OS X, Linux

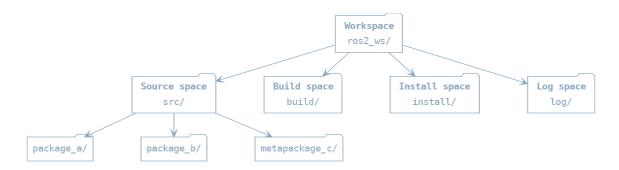
### ROS principles

#### **ROS** workspace



#### Colcon workspace

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



- Source space:
  - Source code of colcon packages
  - Space where you can extract/checkout/clone source code for the packages you want to build
- · Build space
  - · Colcon is invoked here to build packages
  - Colcon and CMake keep intermediate files here
- Install space:
  - Each package will be installed here; by default each package will be installed into a separate subdirectory
- Log space:
  - Contains various logging information about each colcon invocation

### 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 dependencies

After cloning a new package, use the following command to install depenencies:

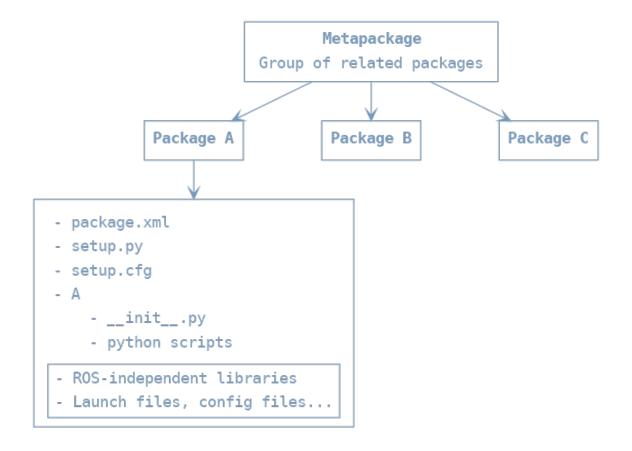
rosdep install --from-paths src --ignore-src -r -y

#### **ROS** package

- Main unit to organize software in ROS
- Buildable and redistributable unit of ROS code
- Consists of (in the case of Python packages):
  - package.xml file containing meta information about the package
    - name
    - version
    - description
    - dependencies
    - etc.
  - setup.py containing instructions for how to install the package
  - setup.cfg is required when a package has executables, so ros2 run can find them
  - /<package\_name> a directory with the same name as your package, used by ROS 2 tools to find your package, contains init .py
  - Anything else
- ros2 run turtlesim turtlesim node

### CMake

For CMake packages (C++), the package contents will be different.



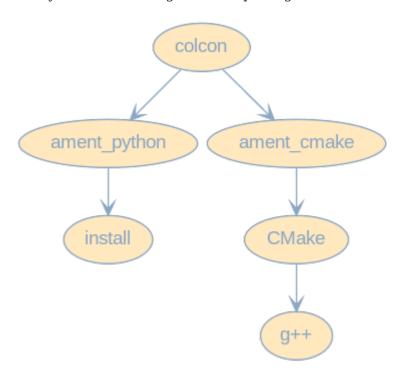
#### **ROS** node

- 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---Colcon**

• System for building software packages in ROS



### **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 ~/ros2\_ws/install/setup.bash

# Gyakorlat

#### 1: Turtlesim

1. Indítsuk el a turtlesim\_node -ot és a turtle\_teleop\_key node-ot az alábbi parancsokkal, külö-külön terminál ablakokban:

ros2 run turtlesim turtlesim node

ros2 run turtlesim turtle\_teleop\_key



Tip

**Terminator**-ban Ctrl-Shift-O , Ctrl-Shift-E billentyű kombinációkkal oszthatjuk tovább az adott ablakot. Ctrl-Shift-W bezárja az aktív ablakot.



Futtatás megszakítása

Ctrl-C

2. Az alábbi ROS parancsok futtatása hasznos információkkal szolgálhat:

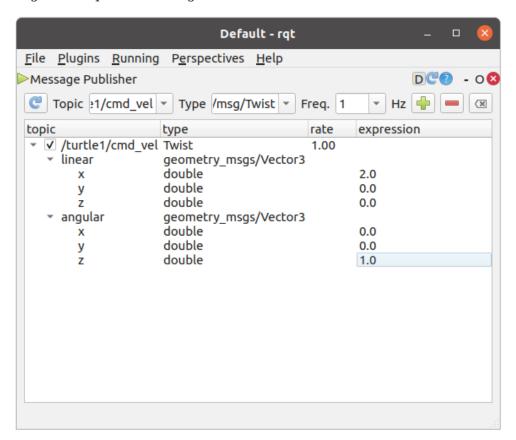
ros2 wtf
ros2 node list
ros2 node info /turtlesim
ros2 topic list
ros2 topic info /turtle1/cmd\_vel
ros2 interface show geometry\_msgs/msg/Twist
ros2 topic echo /turtle1/cmd\_vel

3. Az alábbi paranccsal indítsuk el az rqt\_gui -t:

ros2 run rqt\_gui rqt\_gui

4. Jelenítsük meg a futó node-okat és topic-okat rqt\_gui -ban: Plugins → Introspection → Node Graph.

5. Publikáljunk a /turtle1/cmd\_vel topic-ba szintén az rqt\_gui felhasználásával: Plugins → Topics → Message Publisher.



# 2: ROS 2 workspace létrehozása

1. Hozzunk létre új ROS2 workspace-t ros2\_ws névvel.

```
mkdir -p ~/ros2_ws/src
```

# 3: ROS 2 package létrehozása

1. Hozzunk létre új ROS2 package-et ros2\_course névvel és egy Hello World-del.

```
cd ~/ros2_ws/src
ros2 pkg create --build-type ament_python --node-name hello ros2_course
```



ros2 pkg create --build-type ament python <package name>

2. Build-eljük a workspace-t.

```
cd ~/ros2_ws
colcon build --symlink-install
```

### Symlink

A --symlink-install opció az Install space-be belinkeli a forrás script-eket, így módosítás után nem kell újra build-elnünk.

3. A ~/.bashrc fájl végére illesszük be az alábbi sort:

source ~/ros2\_ws/install/setup.bash

### Importálás QtCreator-ba

New file or project  $\rightarrow$  Other project  $\rightarrow$  ROS Workspace. Válasszuk ki a Colcon-t, mint Build System, és a ros2\_ws-t, mint Worksapce path.

### Importálás CLion-ba

Állítsuk be a Python interpretert Python 3.8-ra, /usr/bin/python3. Adjuk hozzá akövetkező elérési utat: /opt/ros/foxy/lib/python3.8/site-packages. Hozzuk létre a compile commands.json fájlt a ~/ros2 ws/build könyvtárban az alábbi tartalommal:

[

4. Teszteljük a Hello World működését:

ros2 run ros2 course hello

### 4: Publisher implementálása Python-ban

1. Navigáljunk a ros2\_ws/src/ros2\_course/ros2\_course mappába és hozzuk létre a talker.py fájlt az alábbi tartalommal.

```
import rclpy
from rclpy.node import Node
from std msgs.msg import String
class MinimalPublisher(Node):
  def __init__(self):
    super(). init ('minimal publisher')
     self.publisher_ = self.create_publisher(String, 'chatter', 10)
    timer period = 0.5 # seconds
    self.timer = self.create timer(timer period, self.timer callback)
  def timer_callback(self):
     msg = String()
    msg.data = 'Hello World: %d' % self.i
    self.publisher .publish(msg)
    self.get logger().info('Publishing: "%s"' % msg.data)
    self.i += 1
def main(args=None):
  rclpy.init(args=args)
  minimal_publisher = MinimalPublisher()
  rclpy.spin(minimal publisher)
  # Destroy the node explicitly
  # (optional - otherwise it will be done automatically
  # when the garbage collector destroys the node object)
  minimal publisher.destroy node()
  rclpy.shutdown()
if __name__ == '__main__':
  main()
```

2. A setup.py fájlban adjunk meg egy új entry point-on:

```
'talker = ros2_course.talker:main',
```

3. Build-eljük és futtassuk a node-ot:

```
cd ~/ros2_ws
colcon build --symlink-install
ros2 run ros2_course talker
```

4. Ellenőrizzük le a node kimenetét a ros2 topic echo parancs vagy az rqt\_gui használatával.

### 5: Subscriber implementálása Python-ban

1. Navigáljunk a ros2\_ws/src/ros2\_course/ros2\_course mappába és hozzuk létre a listener.py fájlt az alábbi tartalommal.

```
import rclpy
from rclpy.node import Node
from std_msgs.msg import String
class MinimalSubscriber(Node):
  def init (self):
    super(). init ('minimal subscriber')
    self.subscription = self.create subscription(
       String,
       'chatter',
       self.listener_callback,
    self.subscription # prevent unused variable warning
  def listener callback(self, msg):
     self.get logger().info('I heard msg: "%s"' % msg.data)
def main(args=None):
  rclpy.init(args=args)
  minimal_subscriber = MinimalSubscriber()
  rclpy.spin(minimal_subscriber)
  # Destroy the node explicitly
  # (optional - otherwise it will be done automatically
  # when the garbage collector destroys the node object)
  minimal_subscriber.destroy_node()
  rclpy.shutdown()
if __name__ == '__main__':
  main()
```

 $_{2.}\,^{\mathrm{A}}\,$  setup.py fájlban adjunk meg egy új entry point-on:

```
'listener = ros2_course.listener:main',
```

3. Build-eljük és futtassuk mind a 2 node-ot:

```
cd ~/ros2_ws
colcon build --symlink-install
ros2 run ros2_course talker

ros2 run ros2_course listener
```

4. Az rqt\_gui használatával jeleníttessük meg a futó rendszer node-jait és topic-jait:

```
ros2 run rqt_gui rqt_gui
```

# Hasznos linkek

- ROS 2 Tutorials
- What is a ROS 2 package?