



## 02. Linux, ROS alapismeretek



Elmélet

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

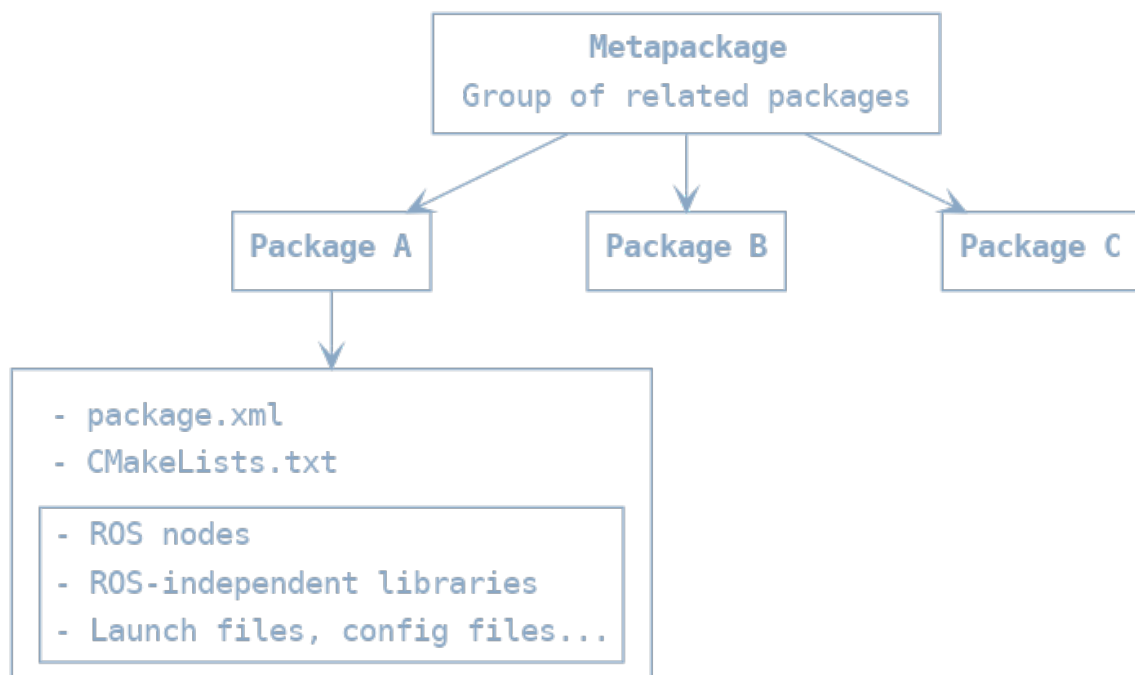
See some basic commands below:

- 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`

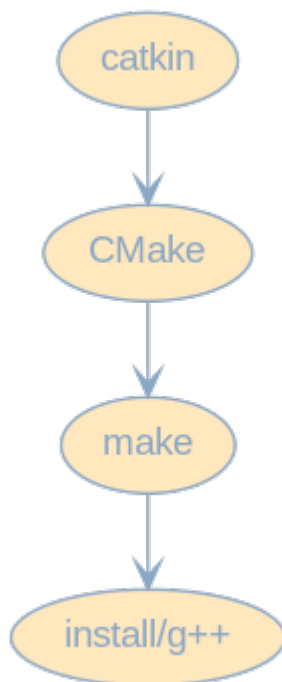
## ROS node

- 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

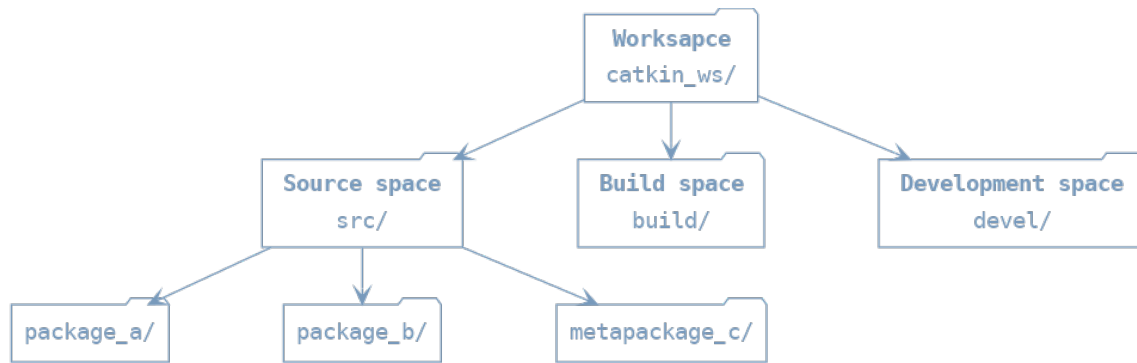
- System for building software packages in ROS



## 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 targets 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

### Figyelem!

Az óra végén a **forráskódokat** mindenkinek fel kell tölteni **Moodle**-re egy zip archívumba csomagolva!

### 1: Turtlesim

1. Indítsuk el a ROS mastert, `turtlesim_node`-ot és a `turtle_teleop_key` node-ot az alábbi parancsokkal, külön-külön terminál ablakokban:

#### Tip

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

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

#### Futtatás megszakítása

```
Ctrl-C
```

2. Az alábbi parancs segítségével jeleníttessük meg a futó rendszer node-jait és topic-jait:



```
roslaunch rqt_graph rqt_graph
```

3. Az alábbi ROS parancsok futtatása hasznos információkkal szolgálhat:

```
roswtf  
rospack list  
rospack find turtlesim  
roslaunch list  
roslaunch info  
roslaunch info /turtlesim  
rostopic list  
rostopic info /turtle1/cmd_vel  
rostopic show geometry_msgs/Twist  
rostopic echo /turtle1/cmd_vel
```

4. Írjuk be a következő parancsot terminálba:

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

## 2: ROS2 workspace létrehozása

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

## 3: ROS2 package létrehozása

1. Hozzunk létre új ROS2 package-et `ros2_course` névvel.

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



### Szintaxis

```
ros2 pkg create --build-type ament_python <package_name>
```

2. Build-eljük a workspace-t.

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

3. A `~/ros2_ws/.bashrc` fájl végére illesszük be az alábbi sort:

```
source ~/ros2_ws/install/setup.bash
```

#### Importálás QtCreator-ba

New file or project -> Other project -> ROS Workspace. Válasszuk ki a Colcon-t, mint Build System, és a `ros2_ws`-t, mint Workspace path.

#### Importálás CLion-ba

Állítsuk be a Python interpretert Python 3.8-ra, `/usr/bin/python3`. Adjuk hozzá a következő 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:

```
[
]
```

1. Teszteljük a Hello World működését:

```
ros2 run ros2_course hello
```

## 4: Publisher implementálása Python-ban

1. Hozzunk létre egy mappát `scripts` névvel a `ros_course` package-ben.

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

2. Navigáljunk a `scripts` mappába és hozzuk létre a `talker.py` fájlt az alábbi tartalommal.

```
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. A CMakeLists.txt -hez adjuk hozzá a következőt:

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

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

```
cd ~/catkin_ws
catkin build
roslaunch ros_course talker.py
```

#### Tip

A node futtatásához szükség van a ROS masterre. Egy külön terminál ablakban indítsuk el a `roscore` paranccsal.

5. Ellenőrizzük le a node kimenetét a `rostopic echo` parancs használatával.

## 5: Subscriber implementálása Python-ban

1. Navigáljunk a `scripts` mappába és hozzuk létre a `listener.py` fájlt az alábbi tartalommal.

```
import rospy
from std_msgs.msg import String

def callback(data):
    print(rospy.get_caller_id() + "I heard %s", 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. A `CMakeLists.txt`-hez adjuk hozzá a következőt:

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

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

```
cd ~/catkin_ws
catkin build
roslaunch ros_course talker.py
```

```
roslaunch ros_course listener.py
```

4. `rqt_graph` használatával jeleníttessük meg a futó rendszer node-jait és topic-jait:

```
roslaunch rqt_graph rqt_graph
```

### **Figyelem!**

Az óra végén a forráskódokat mindenkinek fel kell tölteni Moodle-re egy zip archívumba csomagolva!

## Hasznos linkek

- [ROS Tutorials](#)
- [Curiosity rover simulation](#)
- <https://docs.ros.org/en/galactic/Tutorials/Beginner-Client-Libraries/Creating-A-Workspace/Creating-A-Workspace.html>
- <https://docs.ros.org/en/galactic/Tutorials/Beginner-Client-Libraries/Creating-Your-First-ROS2-Package.html>