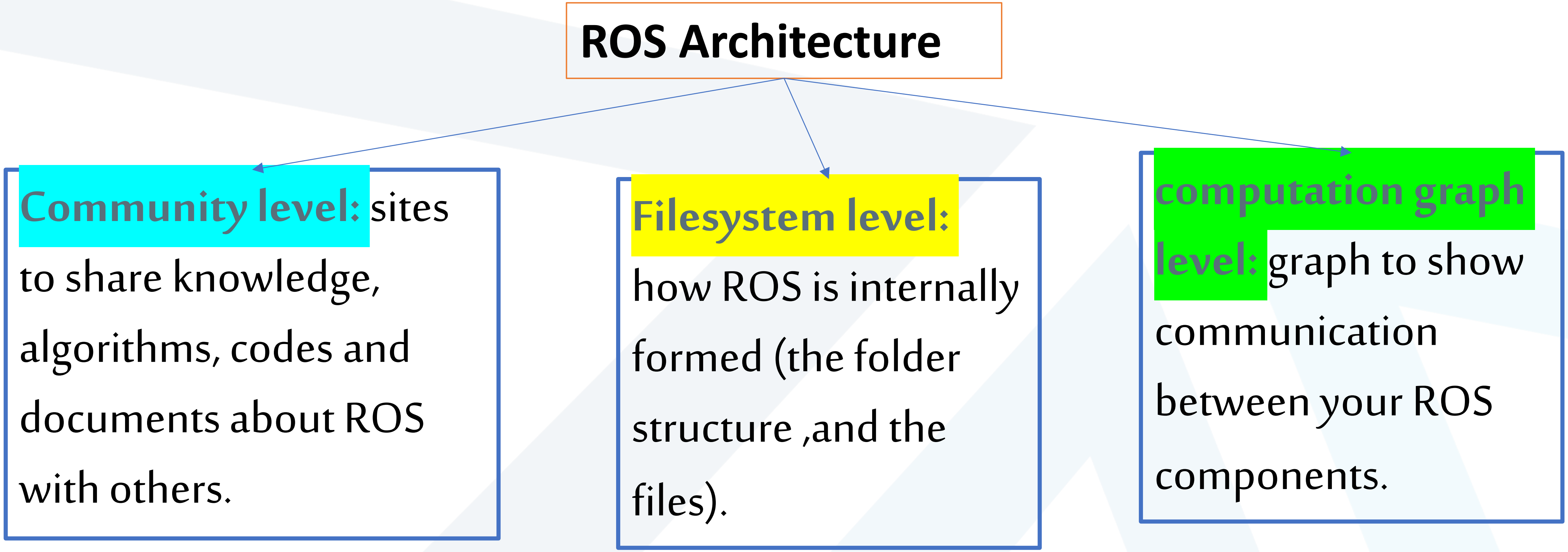
Introduction to Robot Operating System (ROS 1)

ROS architecture (ROS filesystem and ROS Computation Graph level)

Creating and building a ROS workspace and Package

package.xml and package dependencies

Dr. Essa Alghannam



ROS Architecture

Community level: sites to share knowledge, algorithms, codes and documents about ROS with others.

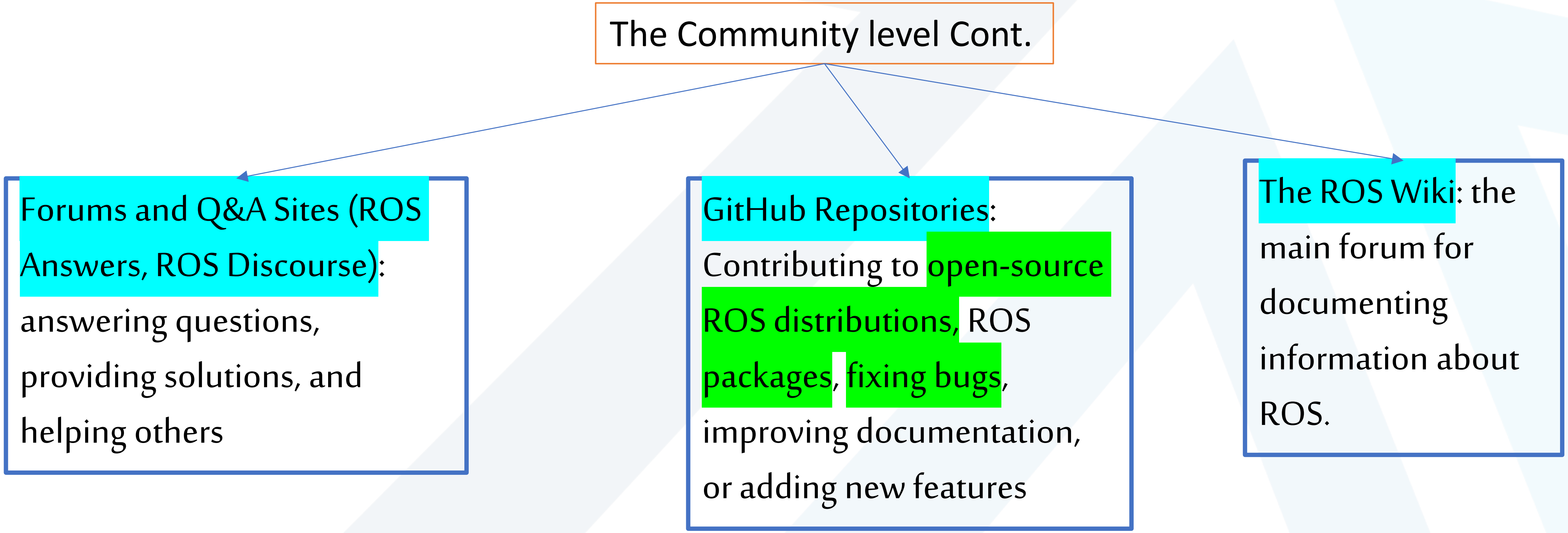
Filesystem level: how ROS is internally formed (the folder structure ,and the files).

computation graph level: graph to show communication between your ROS components.

The Community level

The ROS Community level concepts are ROS resources that enable separate communities to exchange software and knowledge.

These resources include:



The Community level Cont.

Forums and Q&A Sites (ROS Answers, ROS Discourse): answering questions, providing solutions, and helping others

GitHub Repositories: Contributing to open-source ROS distributions, ROS packages, fixing bugs, improving documentation, or adding new features

The ROS Wiki: the main forum for documenting information about ROS.

ROS Filesystem level

Quick Overview of Filesystem Concepts

- **Packages:** Packages are the software organization unit of ROS code (structure and content to create a program within ROS).

  Each package can contain libraries, executables, scripts, or other artifacts.

- **Manifests (package.xml):** A manifest is a description of a *package*. It serves to define dependencies between *packages* and to capture meta information about the *package* like version, maintainer, license, etc.

Package structure

The simplest possible package might have a structure which looks like this:

```
my_package/
  CMakeLists.txt
  package.xml
```

workspace structure:

Packages in a catkin Workspace:

```
workspace_folder/        -- WORKSPACE
src/                     -- SOURCE SPACE
  CMakeLists.txt         -- 'Toplevel' CMake file, provided by catkin
  package_1/
    CMakeLists.txt       -- CMakeLists.txt file for package_1
    package.xml          -- Package manifest for package_1
  ...
  package_n/
    CMakeLists.txt       -- CMakeLists.txt file for package_n
    package.xml          -- Package manifest for package_n
```

Important note

Imagine you want to build a robot navigation system in your workspace. You might create a package called `my_navigation_system` within your workspace. This package might depend on various packages from the `ros-noetic-navigation` metapackage.

`ros-noetic-navigation` is installed system-wide and provides the essential navigation tools.

`my_navigation_system` is in your workspace and leverages those tools to implement your specific navigation system.

Workspace Dependencies:

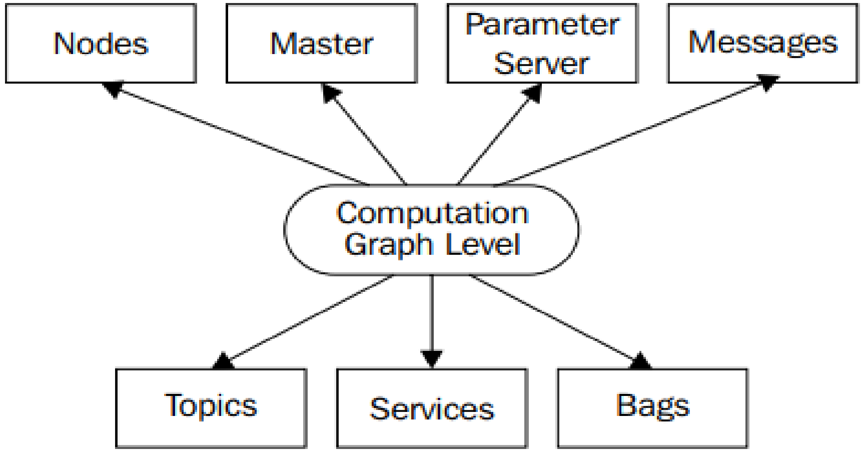
1- When you build packages within your workspace, your package's `package.xml` file will specify its dependencies.

2- If you need to use packages that are part of a metapackage, you would list those packages as dependencies in your `package.xml`.

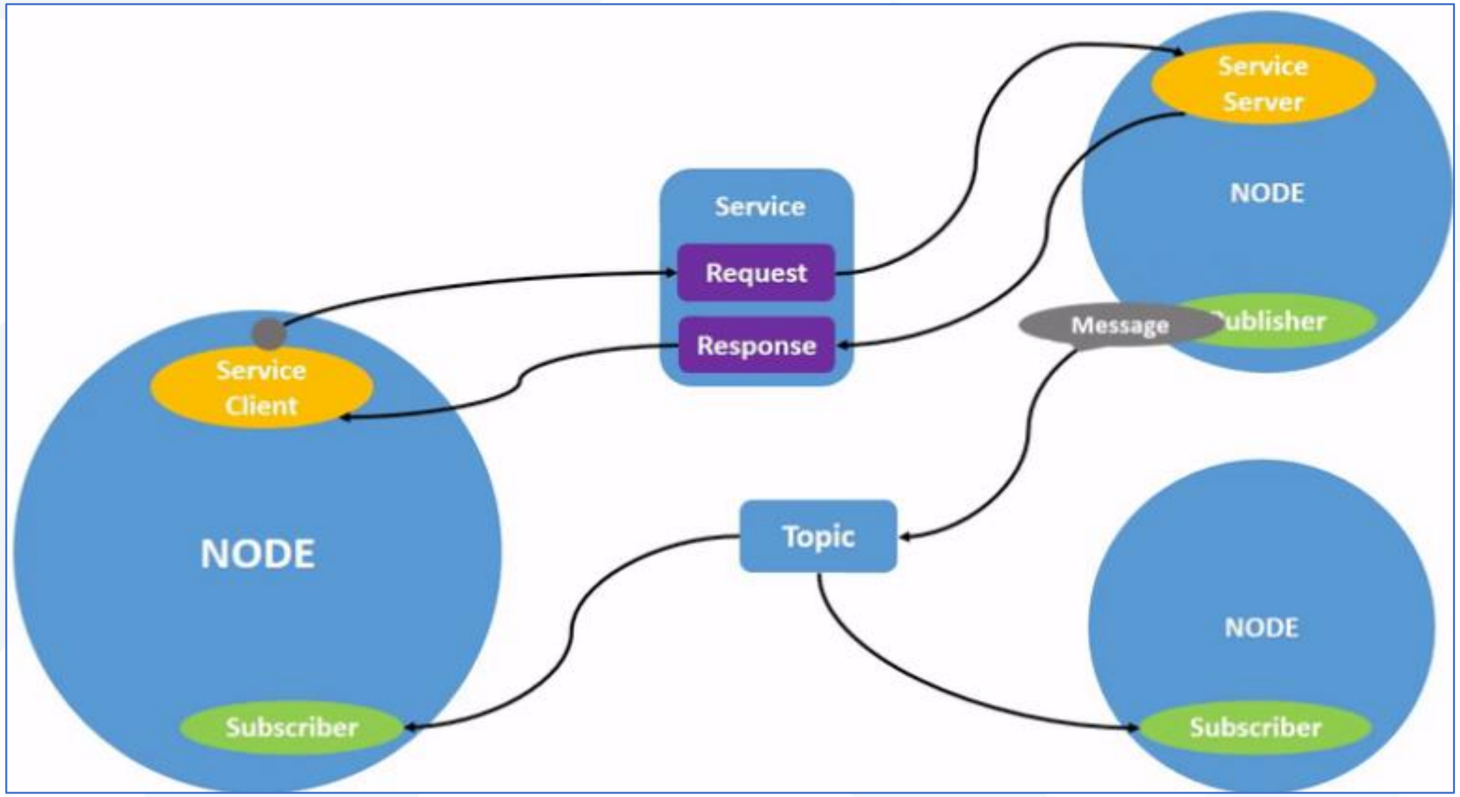
3- The build process will then ensure that those packages (from the metapackage) are available for your workspace.

- System-Wide Installation: Metapackages are typically installed system-wide as part of your ROS distribution.
- Workspace for Your Projects: Use your workspace to develop and manage your custom packages.
- Dependencies in `package.xml`: Specify your package's dependencies in its `package.xml` file.

The ROS Computation Graph level



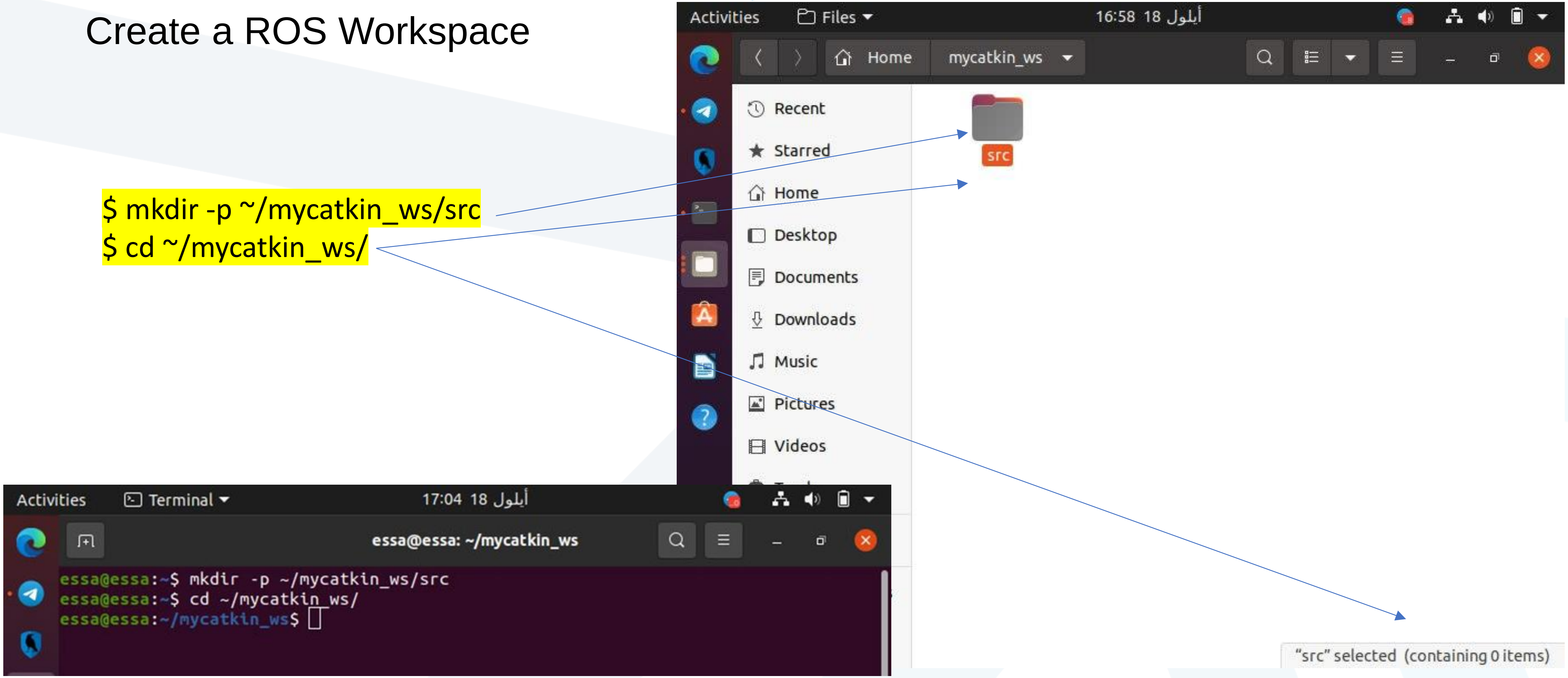
- ROS creates a network where all the processes are connected.
- Any node in the system can access this network, interact with other nodes.
- The basic concepts in this level are nodes, Master, Parameter Server, messages ,services, topics, and bags
- All provide data to the graph in different ways.



Creating a ROS workspace and Package

Create a ROS Workspace

$ mkdir -p ~/mycatkin_ws/src
$ cd ~/mycatkin_ws/



```
essa@essa:~$ mkdir -p ~/mycatkin_ws/src
essa@essa:~$ cd ~/mycatkin_ws/
essa@essa:~/mycatkin_ws$
```

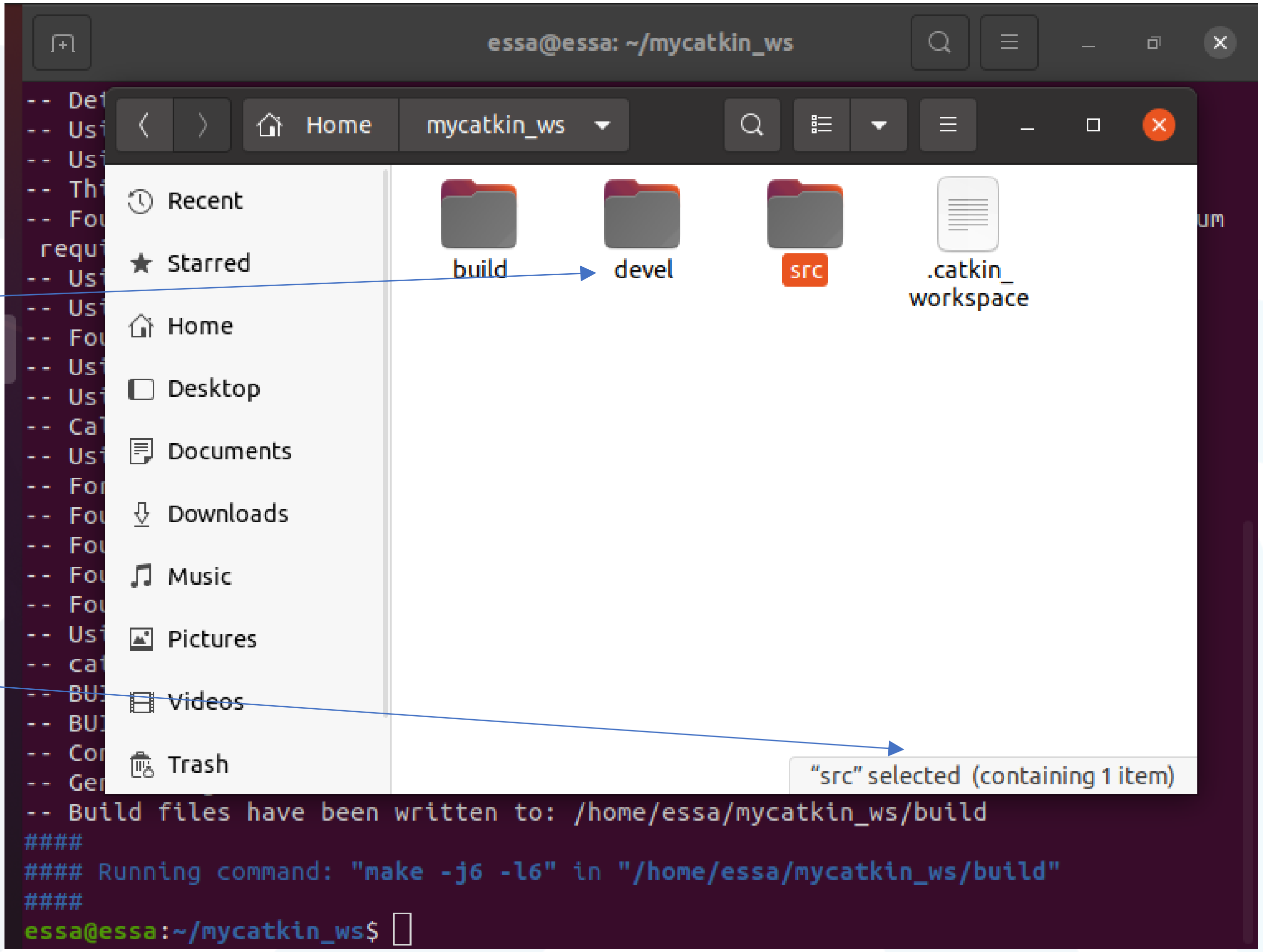
"src" selected (containing 0 items)

Building a catkin workspace before creation of a package

$ catkin_make

- The new build system for ROS is "catkin", while "rosbuild" is the old ROS build system which was replaced by catkin.

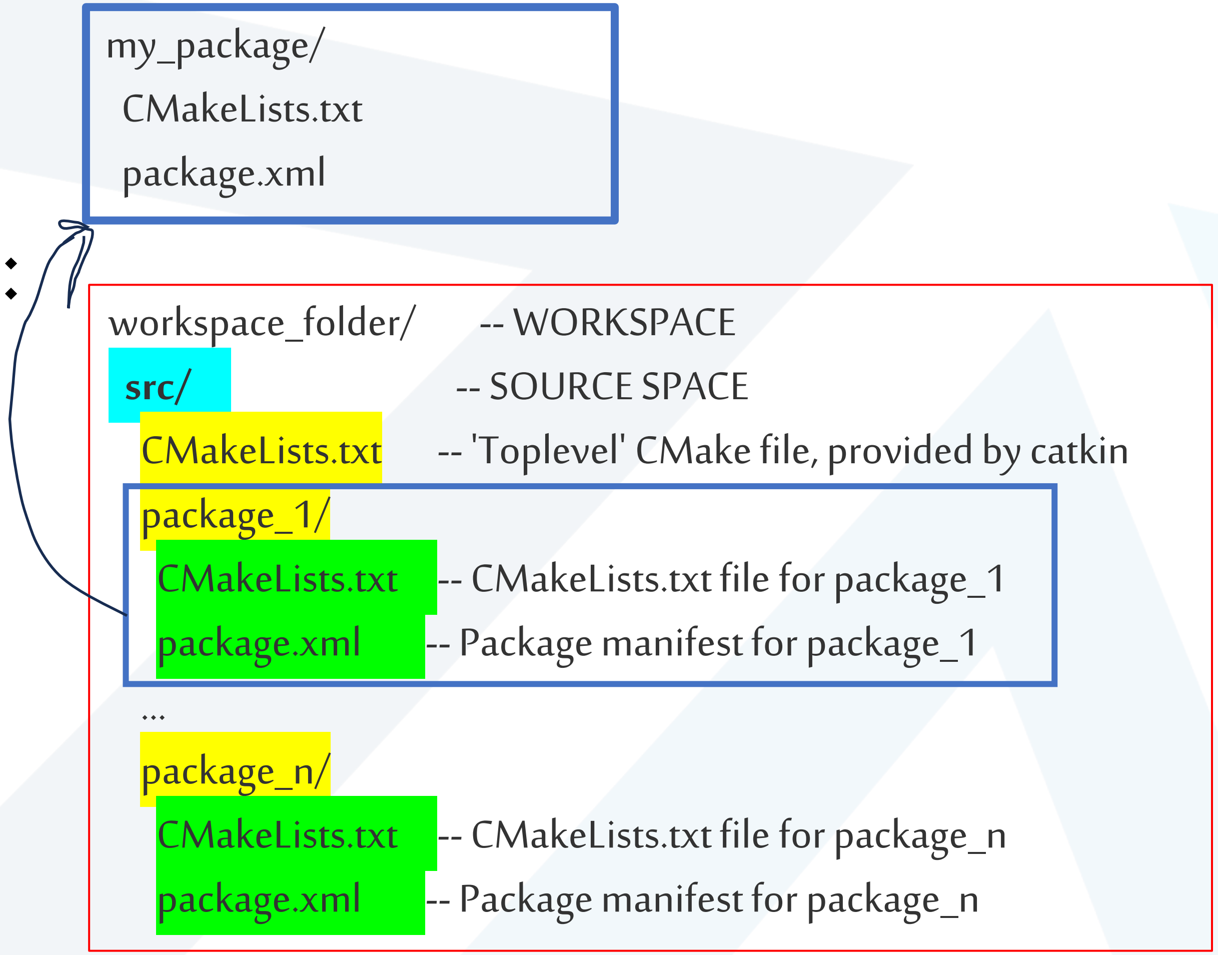
- you should now have a **'build' and 'devel' folder.**
- Inside the **'devel' folder** you can see that there are now **several setup.*sh files.**
- Sourcing any of these files will **overlay this workspace on top of your environment.**
- **CMakeLists.txt** is created in in your **'src' folder**.



```
-- Det
-- Usi
-- Usi
-- Thi
-- Fou
 requ
-- Usi
-- Usi
-- Fou
-- Usi
-- Usi
-- Cal
-- Usi
-- For
-- Fou
-- Fou
-- Fou
-- Fou
-- Usi
-- cat
-- BUI
-- BUI
-- Con
-- Gen
-- Build files have been written to: /home/essa/mycatkin_ws/build
####
#### Running command: "make -j6 -l6" in "/home/essa/mycatkin_ws/build"
####
essa@essa:~/mycatkin_ws$
```

Package structure

The simplest possible package might have a structure which looks like this:



```
my_package/
  CMakeLists.txt
  package.xml
```

workspace structure:

Packages in a catkin Workspace:

```
workspace_folder/        -- WORKSPACE
  src/                   -- SOURCE SPACE
    CMakeLists.txt       -- 'Toplevel' CMake file, provided by catkin
    package_1/
      CMakeLists.txt     -- CMakeLists.txt file for package_1
      package.xml        -- Package manifest for package_1
    ...
    package_n/
      CMakeLists.txt     -- CMakeLists.txt file for package_n
      package.xml        -- Package manifest for package_n
```

Creating a ROS Package

- The package must contain a catkin compliant package.xml file.
- That package.xml file provides meta information about the package.
- The package must contain a CMakeLists.txt which uses catkin.
- Each package must have its own folder
- This means no nested packages nor multiple packages sharing the same directory.

Creating a catkin Package

```
# catkin_create_pkg <package_name> [depend1] [depend2] [depend3]
```

catkin_create_pkg : this command used to create packages.

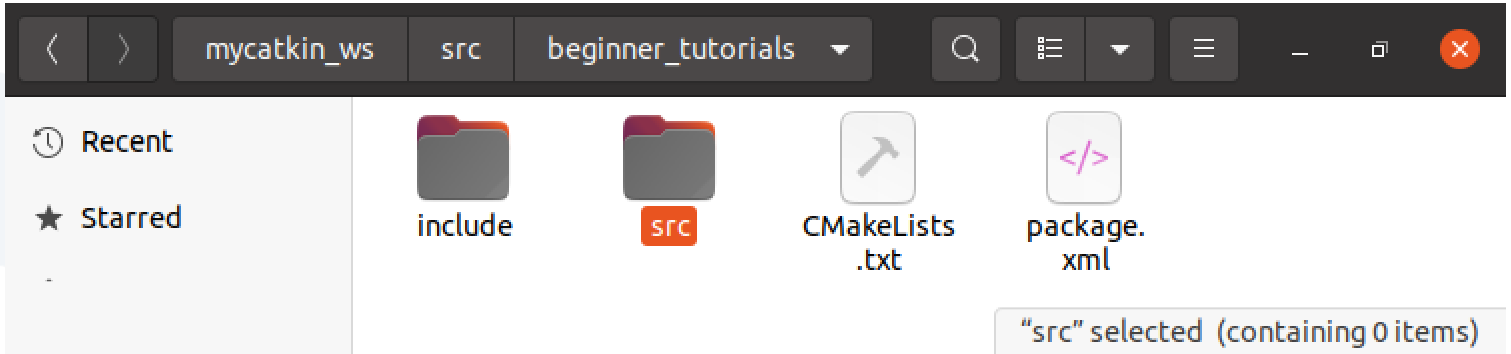
# You should have created this in the Creating a Workspace:

1 `$ cd ~/catkin_ws/src`

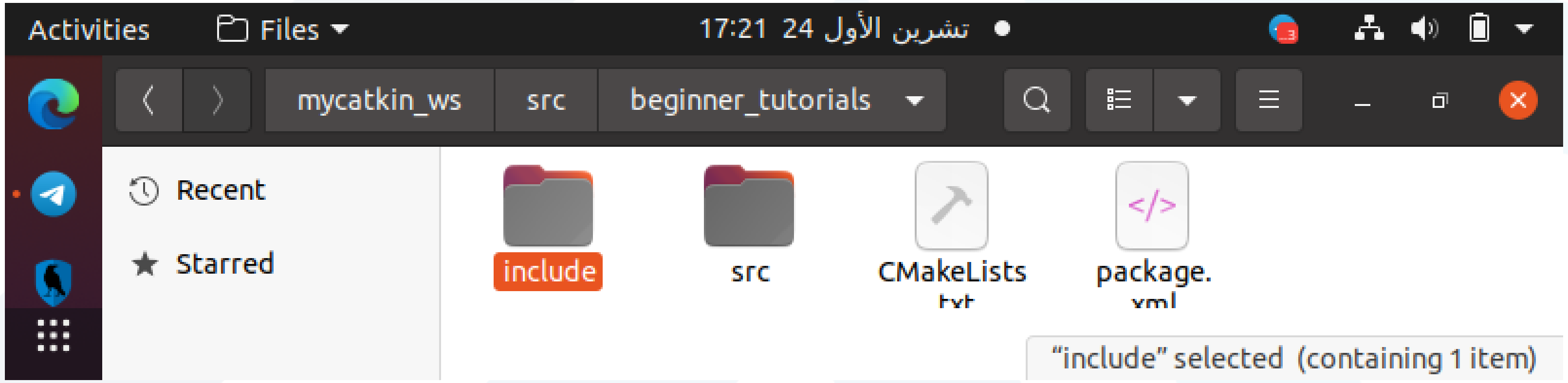
2 `$ catkin_create_pkg beginner_tutorials std_msgs rospy roscpp`

Packages

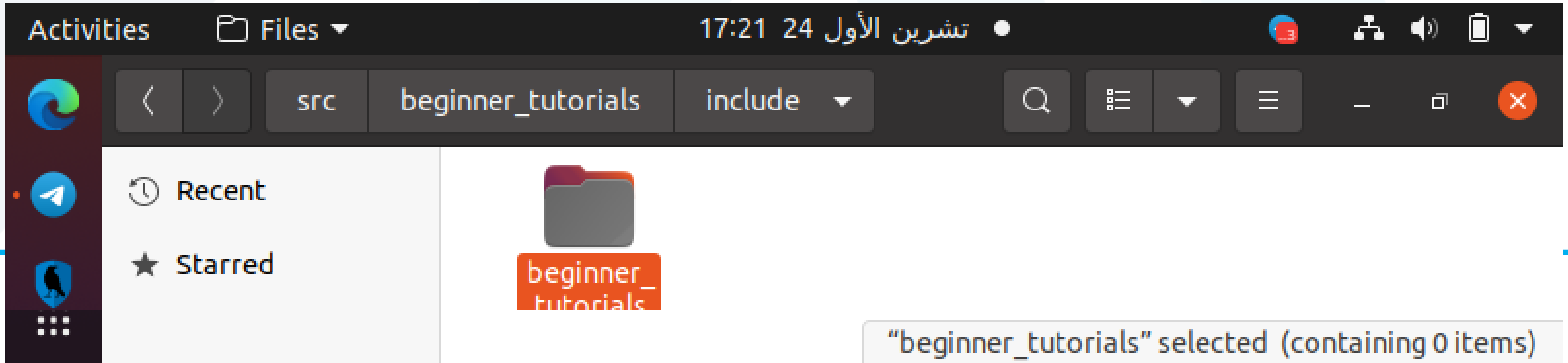
- **include/package_name/:** This directory includes the headers of libraries that you would need.
- **src/: This is where the source files of your programs are present.**
- **CMakeLists.txt:** This is the CMake build file.
- **manifest.xml: This is the package manifest file.**



“src” selected (containing 0 items)



“include” selected (containing 1 item)



“beginner_tutorials” selected (containing 0 items)

Building a catkin workspace after creation of a package

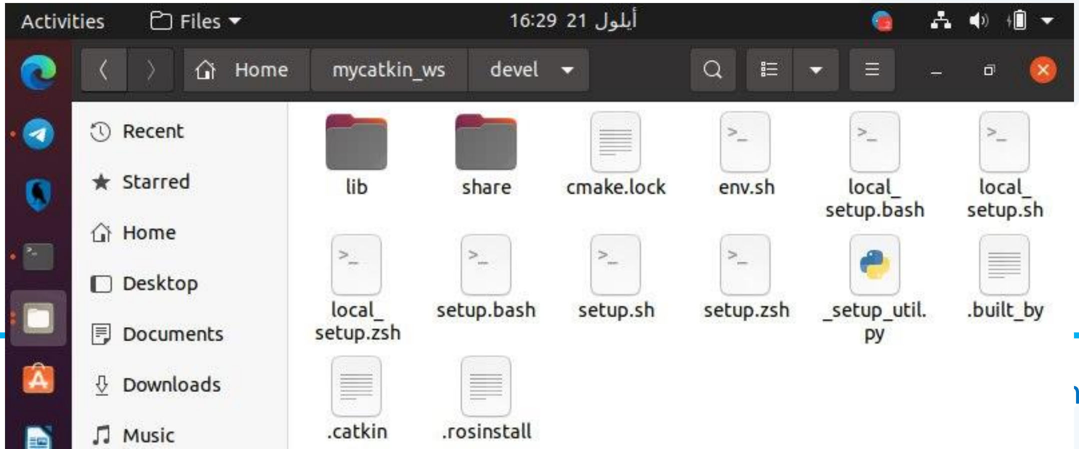
```
$ cd ~/catkin_ws
$ catkin_make
```

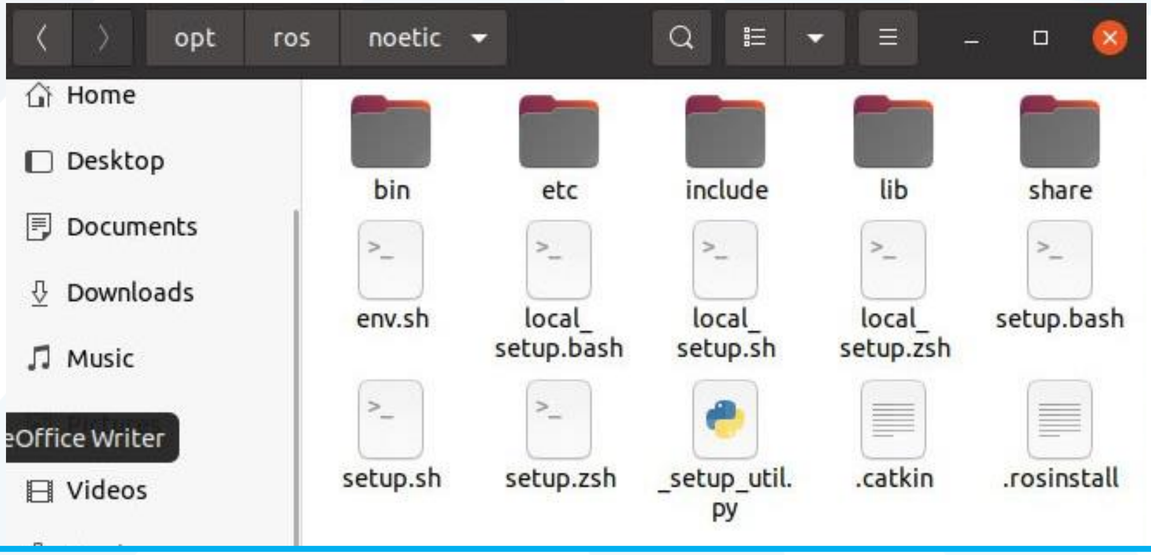
output

```
Base path: /home/essa/mycatkin_ws
Source space: /home/essa/mycatkin_ws/src
Build space: /home/essa/mycatkin_ws/build
Devel space: /home/essa/mycatkin_ws/devel
Install space: /home/essa/mycatkin_ws/install
Creating symlink "/home/essa/mycatkin_ws/src/CMakeLists.txt" pointing to
"/opt/ros/noetic/share/catkin/cmake/toplevel.cmake"
```

catkin_make` compiles your packages but doesn't install them.
To install packages into the `install` folder, you need to run `catkin_make install `

After the workspace has been built it has created a similar structure in the **devel subfolder** as you usually find under /opt/ros/$ROSDISTRO_NAME.





Details of catkin Output:

```
####
#### Running command: "cmake /home/essa/mycatkin_ws/src
-DCATKIN_DEVEL_PREFIX=/home/essa/mycatkin_ws/devel
 -DCMAKE_INSTALL_PREFIX=/home/essa/mycatkin_ws/install -G Unix Makefiles" in "/home/essa/mycatkin_ws/build"
####
-- Using CATKIN_DEVEL_PREFIX: /home/essa/mycatkin_ws/devel
-- Using CMAKE_PREFIX_PATH: /opt/ros/noetic
-- This workspace overlays: /opt/ros/noetic
-- Found PythonInterp: /usr/bin/python3 (found suitable version "3.8.10", minimum required is "3")
-- Using PYTHON_EXECUTABLE: /usr/bin/python3
-- Using Debian Python package layout
-- Using empy: /usr/lib/python3/dist-packages/em.py
-- Using CATKIN_ENABLE_TESTING: ON
-- Call enable_testing()
-- Using CATKIN_TEST_RESULTS_DIR: /home/essa/mycatkin_ws/build/test_results
-- Forcing gtest/gmock from source, though one was otherwise available.
-- Found gtest sources under '/usr/src/googletest': gtests will be built
-- Found gmock sources under '/usr/src/googletest': gmock will be built
```

```
-- Found PythonInterp: /usr/bin/python3 (found version "3.8.10")
-- Using Python nosetests: /usr/bin/nosetests3
-- catkin 0.8.10
-- BUILD_SHARED_LIBS is on
-- BUILD_SHARED_LIBS is on
--
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~
--   traversing 1 packages in topological order:
--   - beginner_tutorials
--
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~
-- +++ processing catkin package: 'beginner_tutorials'
-- ==> add_subdirectory(beginner_tutorials)
-- Configuring done
-- Generating done
-- Build files have been written to: /home/essa/mycatkin_ws/build
####
#### Running command: "make -j6 -l6" in
"/home/essa/mycatkin_ws/build"
####
```

Workspace Folders

1. `src` (Source):

Purpose: This is where you store all the source code for your ROS packages.

Structure: You'll create individual package folders within `src`.

each package folder: contains a `package.xml` (package description file), cmake and the actual source code (C++, Python, etc.).

Example:

* `src/my_package`

* `src/my_other_package`

Workspace Folders

2. `build` (Build):

Purpose: This directory is created by ROS when you run the `catkin_make` command (or `colcon build` for ROS2). It's a temporary working space where ROS builds your packages from the source code in `src`.

Contents: `build` contains intermediate files and compiled code related to your packages during the build process. It's generally not meant to be directly interacted with.

Clean-up: You can safely delete the `build` directory to free up space after a successful build.

Workspace Folders

3. `devel` (Development):

Purpose: This directory holds the compiled, ready-to-use ROS packages after a successful build.

Contents:

1- Include directories: Header files needed by your packages and other ROS packages.

2- Library directories: Compiled libraries used by your ROS nodes and other tools.

3- Executable files: The actual nodes, tools, and scripts you've created within your packages.

Essential for Running: The `devel` directory must be added to your ROS environment's `ROS_PACKAGE_PATH` variable so that ROS knows where to find your compiled packages.

Workspace Folders

In Summary:

1. `src`: Where you write your ROS package code.

2. `build`: A temporary space used by ROS during package compilation.

3. `devel`: Contains the final, compiled packages that ROS uses at runtime.

Important Notes:

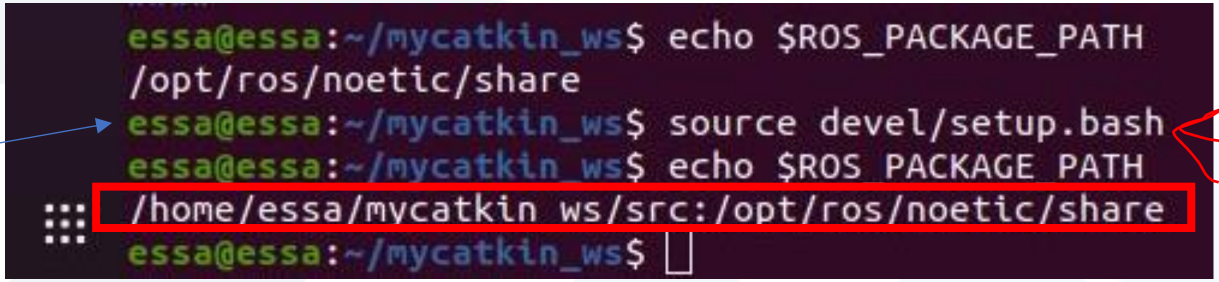
1- `catkin_make` (or `colcon build` in ROS2): The build process uses this command to compile your ROS packages, creating the `build` and `devel` directories.

2- Environment Setup: You usually need to source the `devel/setup.bash` script (or similar) to ensure your ROS environment is correctly configured to use your compiled packages.

ROS_PACKAGE_PATH environment variable and sourcing the setup file

ROS_PACKAGE_PATH environment variable refers to main one before sourcing the new setup .sh

Before continuing source your new setup.*sh file:



```
essa@essa:~/mycatkin_ws$ echo $ROS_PACKAGE_PATH
/opt/ros/noetic/share
essa@essa:~/mycatkin_ws$ source devel/setup.bash
essa@essa:~/mycatkin_ws$ echo $ROS_PACKAGE_PATH
/home/essa/mycatkin ws/src:/opt/ros/noetic/share
essa@essa:~/mycatkin_ws$
```

To make sure your workspace is properly overlayed by the setup script, make sure ROS_PACKAGE_PATH environment variable includes the directory you're in.

To add the workspace to your ROS environment you need to source the generated setup file:

$. ~/mycatkin_ws/devel/setup.bash

Install a catkin workspace

```
catkin_make install
```

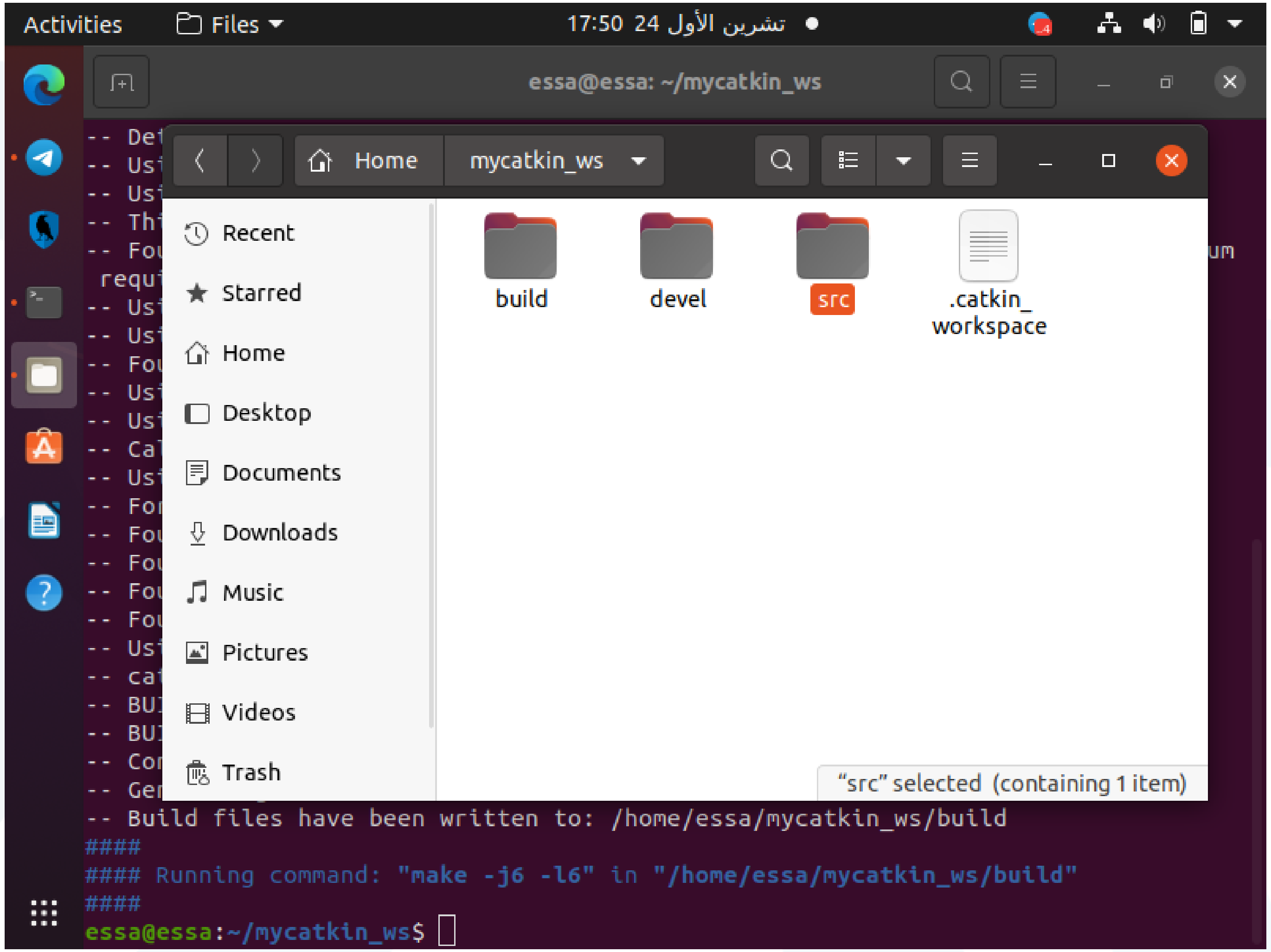
ROS workspace is set up correctly. This means it should contain the following folders:

1- `src`: Where your ROS package code goes.

2- `build`: The build directory.

3- `devel`: Contains the built package code.

4- `install`: Where the packages are installed (after `catkin_make install`).



“src” selected (containing 1 item)

```
-- Det
-- Usi
-- Usi
-- Thi
-- Fou
 requ
-- Usi
-- Usi
-- Fou
-- Usi
-- Usi
-- Ca
-- Usi
-- For
-- Fou
-- Fou
-- Fou
-- Fou
-- Usi
-- cat
-- BUI
-- BUI
-- Cor
-- Ger
-- Build files have been written to: /home/essa/mycatkin_ws/build
####
#### Running command: "make -j6 -l6" in "/home/essa/mycatkin_ws/build"
####
essa@essa:~/mycatkin_ws$ 
```

package.xml and CMakeLists.txt

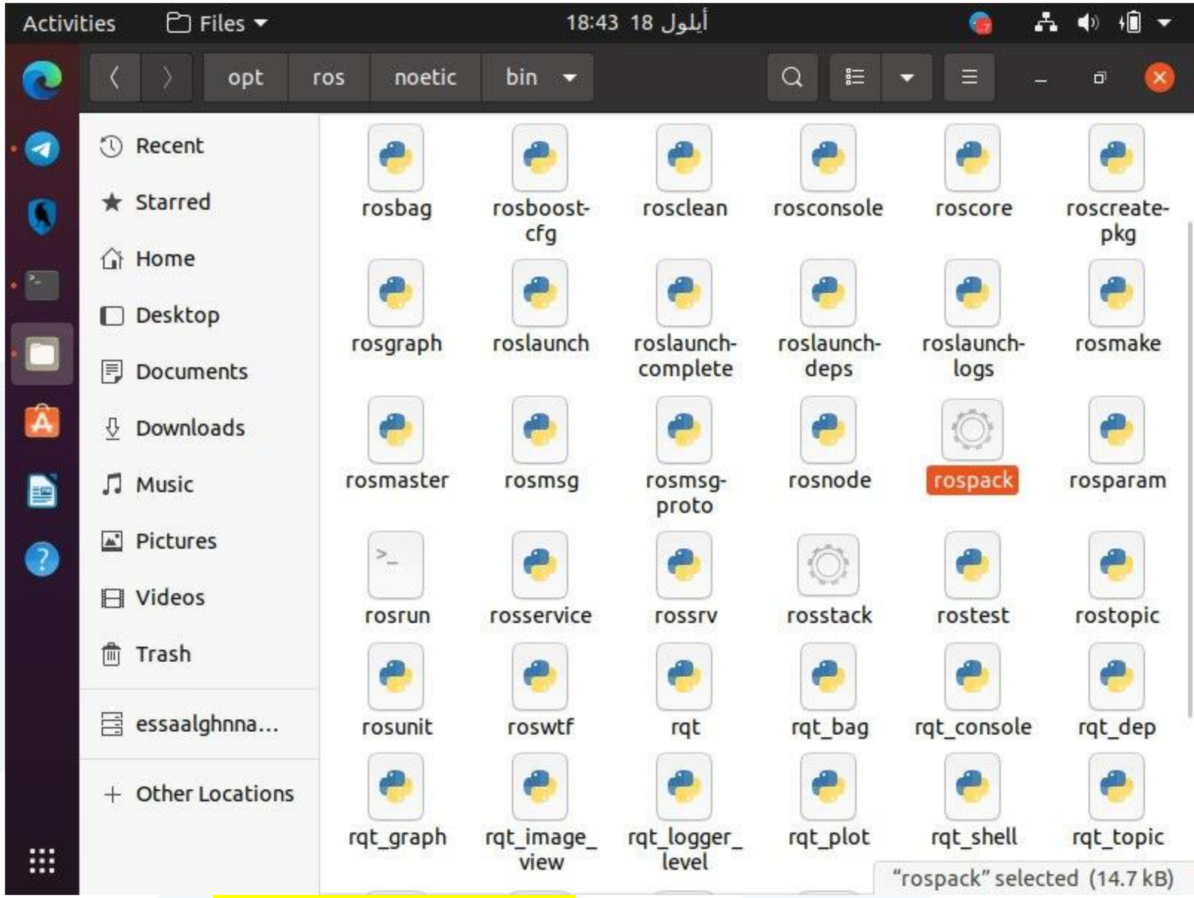
Some ROS Commands

- Navigating with command-line tools such as ls and cd can be very tedious which is why ROS provides tools to help you.

➢ rospack find: To find the path of a package

➢ rosls: This command lists the files from a package.

➢ roscd: This command helps to change the directory.



```
$ rospack find turtlesim
```

To find the path of the turtlesim package

```
opt/ros/noetic/share/turtlesim
```

```
$ rosls turtlesim
```

to list the files inside the pack

```
cmake images srv
package.xml msg
```

- These dependencies for a package are stored in the package.xml file:

```
$ roscd beginner_tutorials
$ cat package.xml
```

cat: Utility to concatenate files to standard output

```
<package format="2">
...
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>rospy</build_depend>
  <build_depend>std_msgs</build_depend>
...
</package>
```

Key Sections of `package.xml`:

1. Package Information:

   * `<name>`: The unique name of your ROS package (e.g., `my_robot_pkg`).

   * `<version>`: The version number (e.g., `1.0.0`).

   * `<description>`: A brief description of what the package does.

2. Dependencies:

   * `<buildtool_depend>`: Lists the build tools necessary to compile the package. Typically includes `catkin` (for building packages with catkin).

   * `<build_depend>`: Lists the build-time dependencies, meaning other ROS packages needed during compilation.

   * `<run_depend>`: Lists the run-time dependencies, meaning other ROS packages needed for the package to execute successfully.

   * `<depend>`: This is a deprecated tag that can be used for both build-time and run-time dependencies. It's usually used for packages that are both build-time and run-time dependent.

3. Maintainers and Authors and url

   * `<maintainer>`: Lists the person or team responsible for the package. It includes their name and email address.

4. License:

   * `<license>`: Specifies the license under which the package is distributed.

```
<?xml version="1.0"?>
<package format="2">
 <name>beginner_tutorials</name>
 <version>0.0.0</version>
 <description>The beginner_tutorials package</description>

 <!-- One maintainer tag required, multiple allowed, one person per tag -->
 <!-- Example:  -->
 <!-- <maintainer email="jane.doe@example.com">Jane Doe</maintainer> -->
 <maintainer email="essa@todo.todo">essa</maintainer>

 <!-- One license tag required, multiple allowed, one license per tag -->
 <!-- Commonly used license strings: -->
 <!--   BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3 -->
 <license>TODO</license>

  <!-- Url tags are optional, but multiple are allowed, one per tag -->
 <!-- Optional attribute type can be: website, bugtracker, or repository -->
 <!-- Example: -->
<url type="website">http://wiki.ros.org/beginner_tutorials</url>

 <!-- Author tags are optional, multiple are allowed, one per tag -->
 <!-- Authors do not have to be maintainers, but could be -->
 <!-- Example: -->
<author email="jane.doe@example.com">Jane Doe</author>
```

```
<!-- The *depend tags are used to specify dependencies -->
<!-- Dependencies can be catkin packages or system dependencies -->
<!-- Examples: -->
<!-- Use depend as a shortcut for packages that are both build and exec dependencies -->
<!--   <depend>roscpp</depend> -->
<!--   Note that this is equivalent to the following: -->
<!--   <build_depend>roscpp</build_depend> -->
<!--   <exec_depend>roscpp</exec_depend> -->
<!-- Use build_depend for packages you need at compile time: -->
<!--   <build_depend>message_generation</build_depend> -->
<!-- Use build_export_depend for packages you need in order to build against this package: -->
<!--   <build_export_depend>message_generation</build_export_depend> -->
<!-- Use buildtool_depend for build tool packages: -->
<!--   <buildtool_depend>catkin</buildtool_depend> -->
<!-- Use exec_depend for packages you need at runtime: -->
<!--   <exec_depend>message_runtime</exec_depend> -->
<!-- Use test_depend for packages you need only for testing: -->
<!--   <test_depend>gtest</test_depend> -->
<!-- Use doc_depend for packages you need only for building documentation: -->
<!--   <doc_depend>doxygen</doc_depend> -->
<buildtool_depend>catkin</buildtool_depend>
<build_depend>roscpp</build_depend>
<build_depend>rospy</build_depend>
<build_depend>std_msgs</build_depend>
<build_export_depend>roscpp</build_export_depend>
<build_export_depend>rospy</build_export_depend>
<build_export_depend>std_msgs</build_export_depend>
<exec_depend>roscpp</exec_depend>
<exec_depend>rospy</exec_depend>
<exec_depend>std_msgs</exec_depend>


<!-- The export tag contains other, unspecified, tags -->
<export>
  <!-- Other tools can request additional information be placed here -->

</export>
</package>
```

```
<?xml version="1.0"?>
<package format="2">
  <name>beginner_tutorials</name>
  <version>0.1.0</version>
  <description>The beginner_tutorials package</description>

  <maintainer email="you@yourdomain.tld">Your Name</maintainer>
  <license>BSD</license>
  <url type="website">http://wiki.ros.org/beginner_tutorials</url>
  <author email="you@yourdomain.tld">Jane Doe</author>

  <buildtool_depend>catkin</buildtool_depend>

  <build_depend>roscpp</build_depend>
  <build_depend>rospy</build_depend>
  <build_depend>std_msgs</build_depend>

  <exec_depend>roscpp</exec_depend>
  <exec_depend>rospy</exec_depend>
  <exec_depend>std_msgs</exec_depend>

</package>
```

1. `<buildtool_depend>catkin</buildtool_depend>`: This specifies that the `catkin` build system is required to build this package. . It's a core ROS tool for package management.
2. `<build_depend>roscpp</build_depend>`: Indicates that the `roscpp` package is needed during the build process. `roscpp` provides C++ libraries for ROS development.
3. `<build_depend>rospy</build_depend>`: Indicates that the `rospy` package is needed during the build process. `rospy` provides Python libraries for ROS development.
4. `<build_depend>std_msgs</build_depend>`: Indicates that the `std_msgs` package is needed during the build process. `std_msgs` provides standard message types (e.g., for integers, floats, strings) used in ROS.
5. `<build_export_depend>roscpp</build_export_depend>`: This indicates that `roscpp` needs to be built before building against this package.
6. `<build_export_depend>rospy</build_export_depend>`: This indicates that `rospy` needs to be built before building against this package.
7. `<build_export_depend>std_msgs</build_export_depend>`: This indicates that `std_msgs` needs to be built before building against this package.
8. `<exec_depend>roscpp</exec_depend>`: This indicates that `roscpp` is needed at runtime.
9. `<exec_depend>rospy</exec_depend>`: This indicates that `rospy` is needed at runtime.
10. `<exec_depend>std_msgs</exec_depend>`: This indicates that `std_msgs` is needed at runtime.

```xml
<?xml version="1.0"?>
<package>
  <name>my_robot_pkg</name>
  <version>1.0.0</version>
  <description>A simple ROS package to control a simulated robot.</description>
  <maintainer email="your_email@example.com">Your Name</maintainer>
  <license>BSD</license>

  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>rospy</build_depend>
  <build_depend>std_msgs</build_depend>
  <run_depend>rospy</run_depend>
  <run_depend>std_msgs</run_depend>
</package>
```

1. `<buildtool_depend>catkin</buildtool_depend>`: This specifies that the `catkin` build system is required to build your package. It's a core ROS tool for package management.
2. `<build_depend>rospy</build_depend>`: This indicates that the `rospy` package is needed during the build process. `rospy` provides Python libraries for ROS development.
3. `<build_depend>std_msgs</build_depend>`: This indicates that the `std_msgs` package is needed during the build process. `std_msgs` provides standard message types (like integers, floats, strings) used in ROS.
4. `<run_depend>rospy</run_depend>`: This indicates that the `rospy` package is required when your package is running (at runtime).
5. `<run_depend>std_msgs</run_depend>`: This indicates that the `std_msgs` package is required when your package is running (at runtime).

CMakeLists.txt

- The CMakeLists.txt file created by catkin_create_pkg will be covered in the later tutorials about building ROS code.

جَـامعة
المَـنارة
MANARA UNIVERSITY

package dependencies

package dependencies

1-First-order dependencies

$ rospack depends1 beginner_tutorials

- roscpp
- rospy
- std_msgs

`rospack` is a command-line tool within ROS used to manage and inspect ROS packages. It's essentially a package manager, helping you understand dependencies and relationships between packages.

rospack: can recursively determine all nested dependencies.

$ rospack depends beginner_tutorials

- cpp_common
- rostime
- roscpp_traits
- roscpp_serialization
- catkin
- genmsg
- genpy
- message_runtime
- gencpp
- geneus
- Gennodejs
- genlisp
- message_generation
- rosbuild
- rosconsole
- std_msgs
- rosgraph_msgs
- xmlrpcpp
- roscpp
- rosgraph
- ros_environment
- rospack
- roslib
- rospy

2-Indirect dependencies

$ rospack depends1 rospy

- genpy
- roscpp
- rosgraph
- rosgraph_msgs
- roslib
- std_msgs

`depends1` is a special directive within the `package.xml` file (a ROS package's descriptor). It declares the dependencies that a package needs to compile and run correctly.

  * The "1" in `depends1` signifies that the package is "build-time" dependent, meaning it's required to build the package itself.

  * There's also `depends`, which indicates "run-time" dependencies, needed for the package to execute properly.

`beginner_tutorials` is the name of a specific ROS package

package dependencies
1-First-order dependencies

1. Calls `rospack`: It tells the ROS system to use the `rospack` tool.

2. Specifies the `depends1` directive: It asks `rospack` to list all the build-time dependencies for the `beginner_tutorials` package.

you'll typically see a list of other packages that `beginner_tutorials` relies on to be built successfully. These dependencies might include:

* Core ROS packages: Like `rospy`, `roscpp`, `std_msgs`, and `sensor_msgs` (for basic communication and message definitions).

* Specific libraries: If `beginner_tutorials` uses external libraries, they might be listed here.

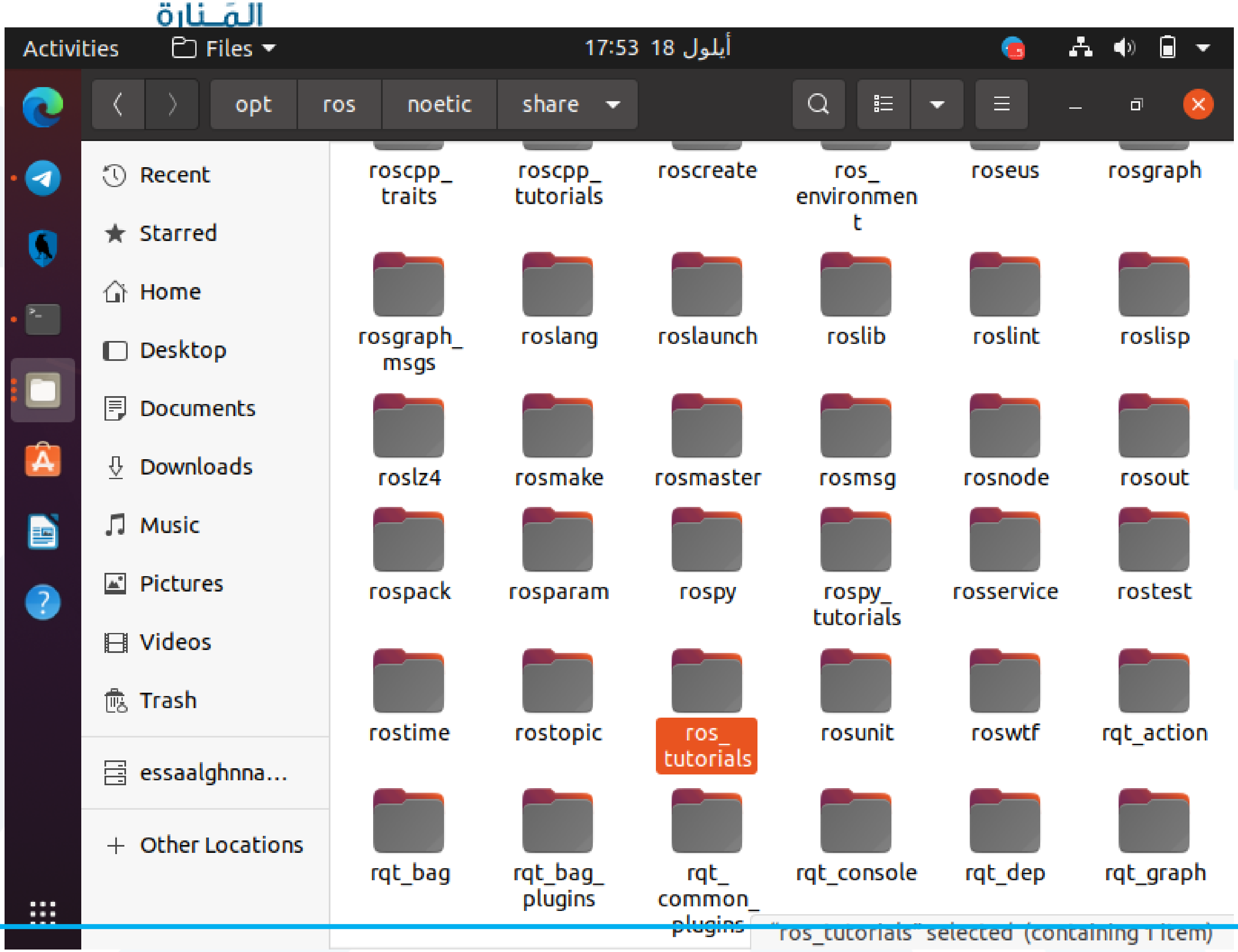
```
sudo apt update
sudo apt upgrade
sudo rosdep init
rosdep update
source /opt/ros/noetic/setup.bash
mkdir -p ~/mycatkin_ws/src
cd ~/mycatkin_ws/
catkin_make
catkin_make install
cd ~/mycatkin_ws/src
catkin_create_pkg beginner_tutorials std_msgs rospy roscpp
cd ~/mycatkin_ws
catkin_make
. ~/mycatkin_ws/devel/setup.bash
catkin_make install
```

source is just a bash builtin that does exactly the same as (.).

Navigating the ROS Filesystem

- we will inspect a package in ros-tutorials, please install it using

$ sudo apt-get install ros-noetic-ros-tutorials



“ros_tutorials” selected (containing 1 item)

جَـامعة
المَـنارة
MANARA UNIVERSITY

شكرا لحسن الاصغاء