F1/10
Autonomous Racing

ROS – Build +
Filesystem and Workspaces

Madhur Behl

Rice Hall 120

ROS Nodes

- Single-purpose, executable program
- Individually compiled, executed, and managed
- Organized in *packages*

Run a node with

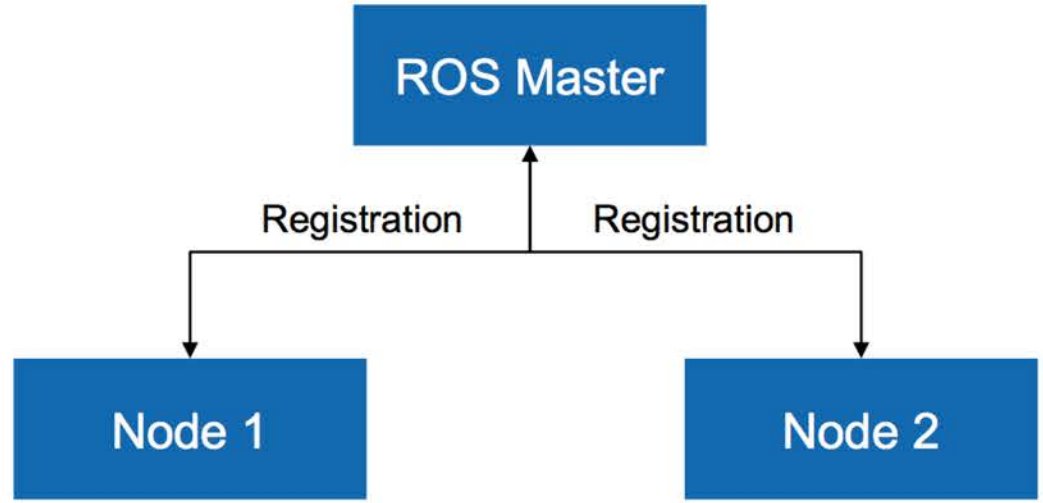
```
> rosrun package_name node_name
```

See active nodes with

```
> rosnode list
```

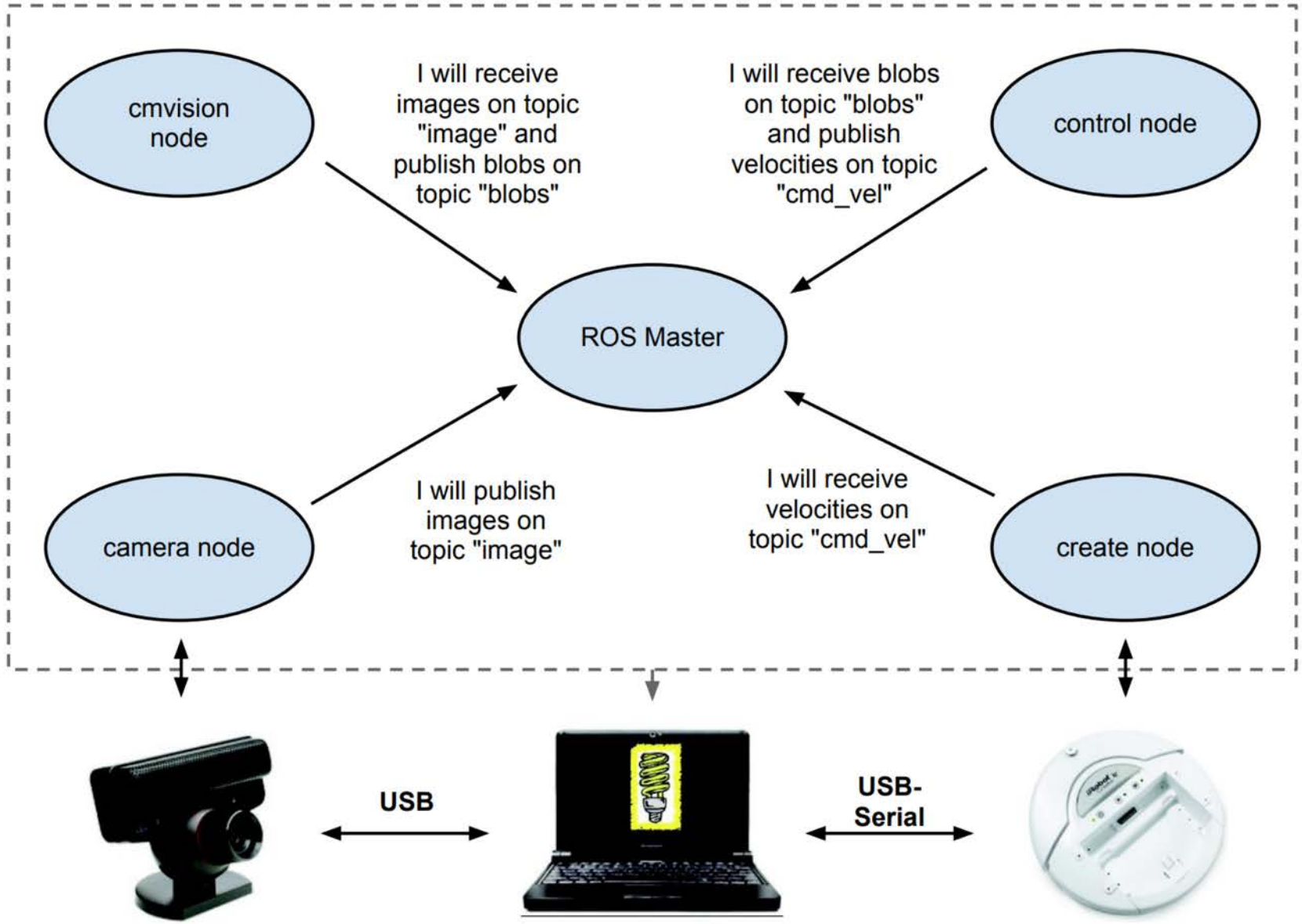
Retrieve information about a node with

```
> rosnode info node_name
```

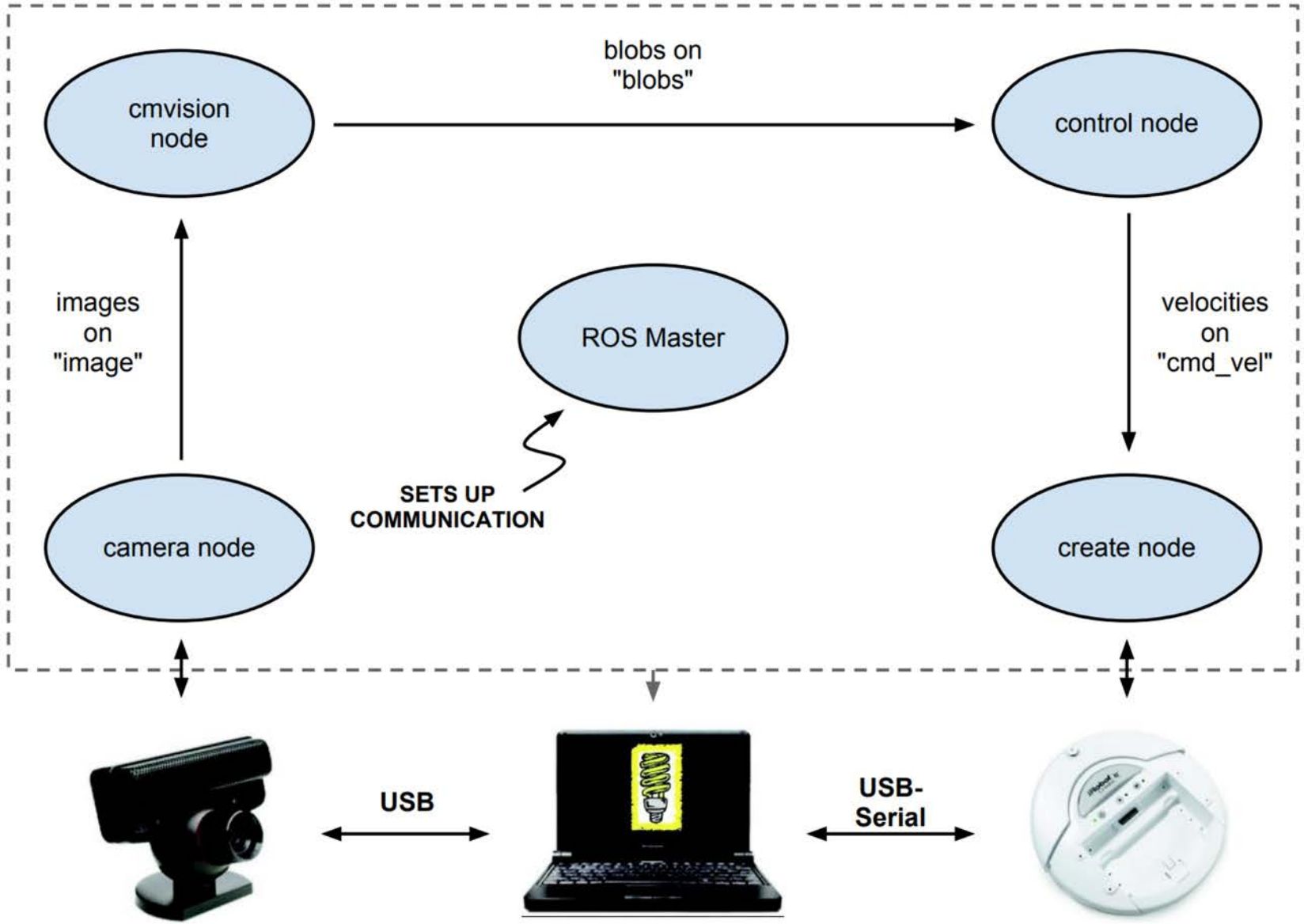


More info
http://wiki.ros.org/rosnode

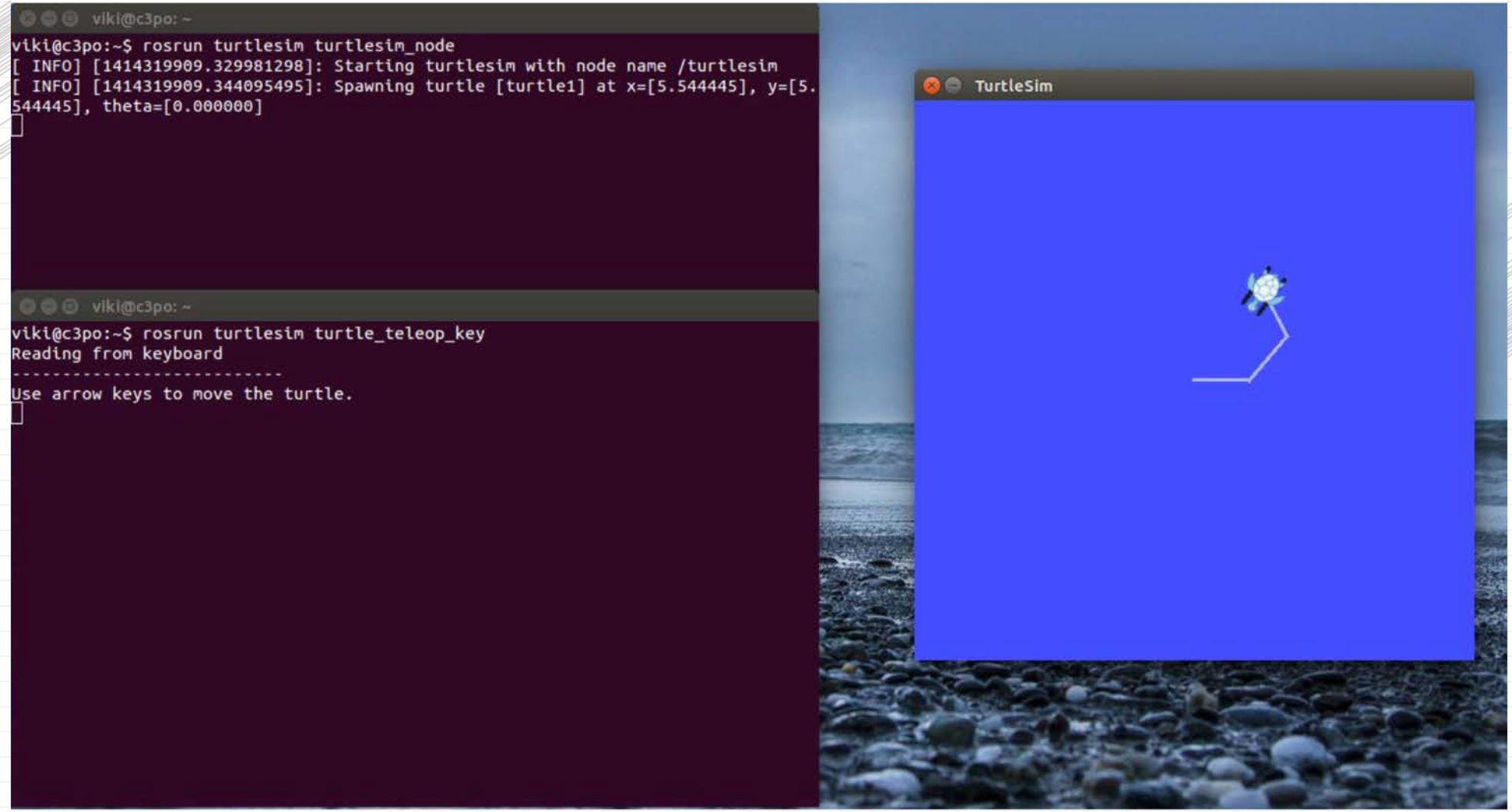
Review - How ROS works



Review - How ROS works



- **Demo – Turtlesim:** In separate terminals, run:
  - roscore
  - rosrun turtlesim turtlesim_node
  - rosrun turtlesim turtle_teleop_key



rosnode info

```
viki@c3po: ~
viki@c3po:~$ rosnode info turtlesim
--------------------------------------------------------------------------------
Node [/turtlesim]
Publications:
 * /turtle1/color_sensor [turtlesim/Color]
 * /rosout [rosgraph_msgs/Log]
 * /turtle1/pose [turtlesim/Pose]

Subscriptions:
 * /turtle1/cmd_vel [geometry_msgs/Twist]

Services:
 * /turtle1/teleport_absolute
 * /turtlesim/get_loggers
 * /turtlesim/set_logger_level
 * /reset
 * /spawn
 * /clear
 * /turtle1/set_pen
 * /turtle1/teleport_relative
 * /kill


contacting node http://c3po:54205/ ...
Pid: 3825
Connections:
 * topic: /rosout
    * to: /rosout
    * direction: outbound
    * transport: TCPROS
 * topic: /turtle1/cmd_vel
    * to: /teleop_turtle (http://c3po:47526/)
    * direction: inbound
    * transport: TCPROS

viki@c3po:~$
```

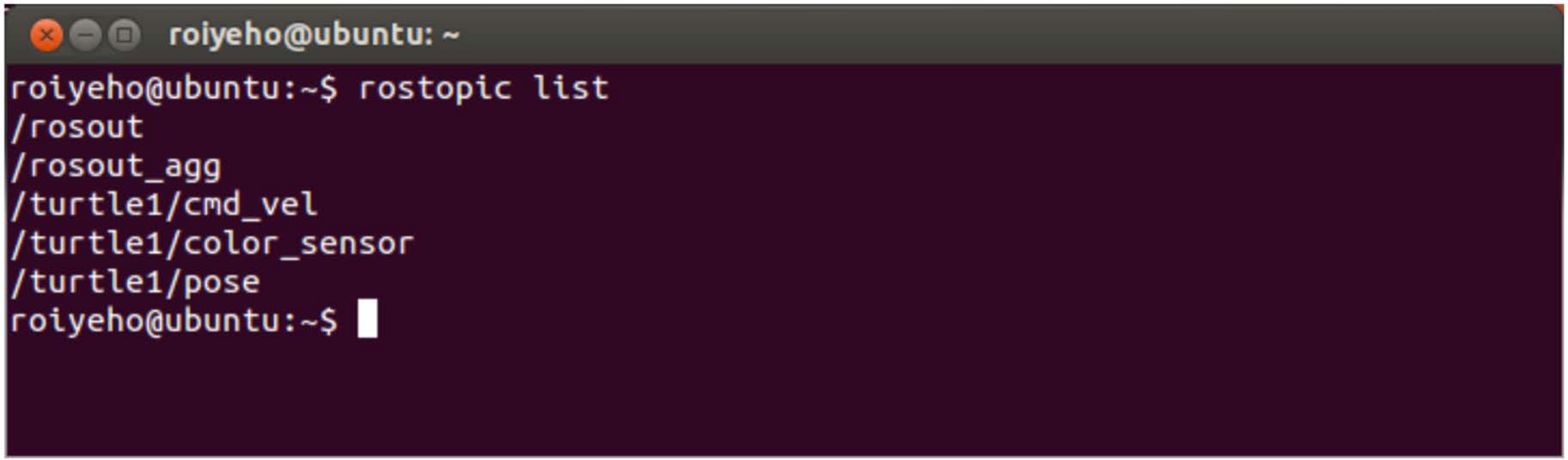
rostopic

- Gives information about a topic and allows to publish messages on a topic

Command	
$rostopic list	List active topics
$rosnode echo /topic	Prints messages of the topic to the screen
$rostopic info /topic	Print information about a topic
$rostopic type /topic	Prints the type of messages the topic publishes
$rostopic pub /topic type args	Publishes data to a topic

rostopic list

- Displays the list of current topics:



Publish to ROS Topic

- Use the **rostopic pub** command to publish messages to a topic
- For example, to make the turtle move forward at a 0.2m/s speed, you can publish a cmd_vel message to the topic /turtle1/cmd_vel:

```
$ rostopic pub /turtle1/cmd_vel geometry_msgs/Twist '{linear: {x: 0.2, y: 0, z: 0}, angular: {x: 0, y: 0, z: 0}}'
```

- To specify only the linear x velocity:

```
$ rostopic pub /turtle1/cmd_vel geometry_msgs/Twist '{linear: {x: 0.2}}'
```

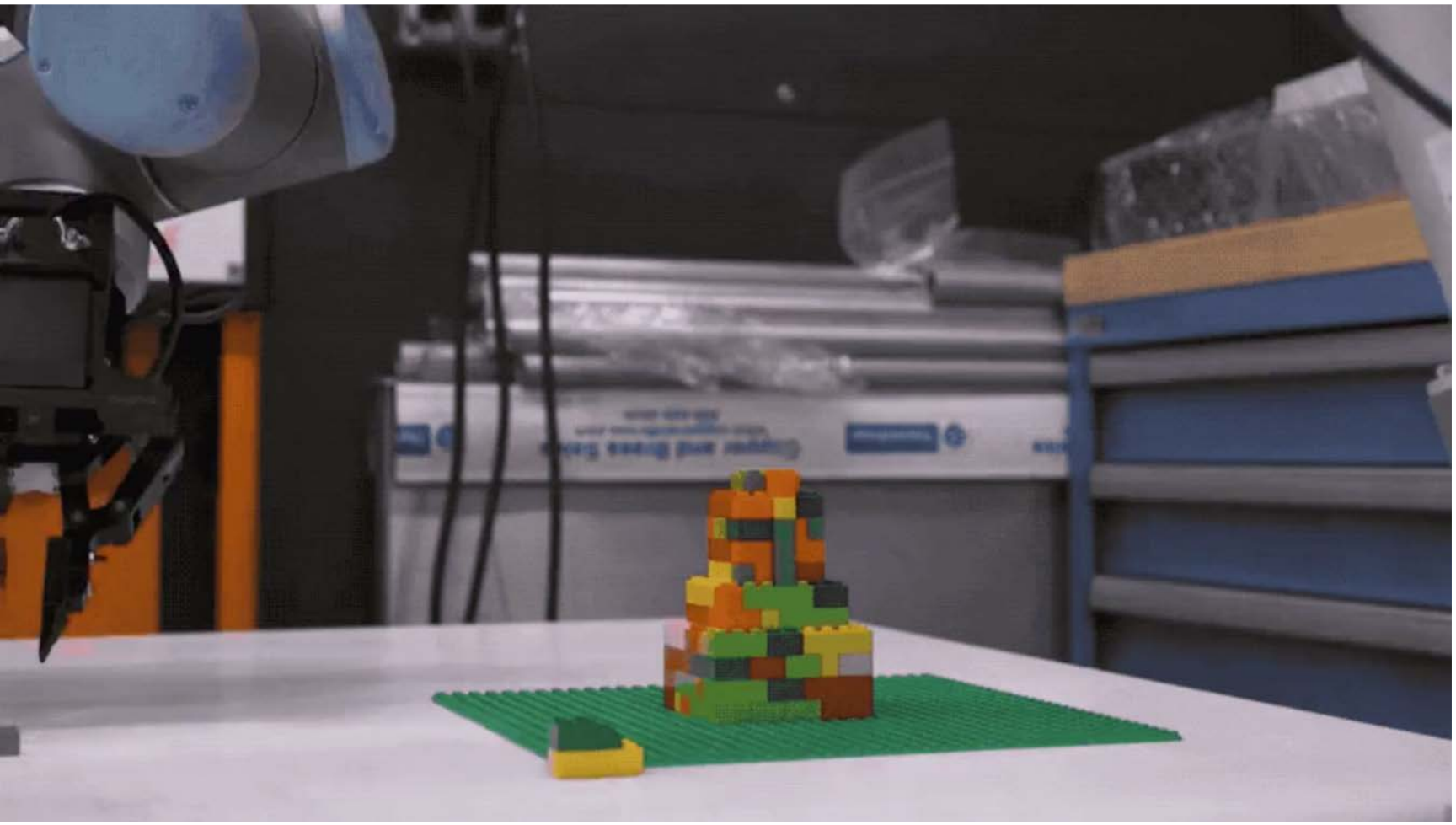
Topic Type Arguments

Proper way to terminate ROS nodes

rosnode kill *node_name*

You can also kill a node using the usual Ctrl-C *technique. However, that method may not give the node a chance to unregister itself from the master. A symptom of this problem is that the killed node may still be listed by* `rosnode list` *for a while. This is harmless, but might make it more difficult to tell what's going on. To remove dead nodes from the list, you can use this command:*

```
rosnode cleanup
```

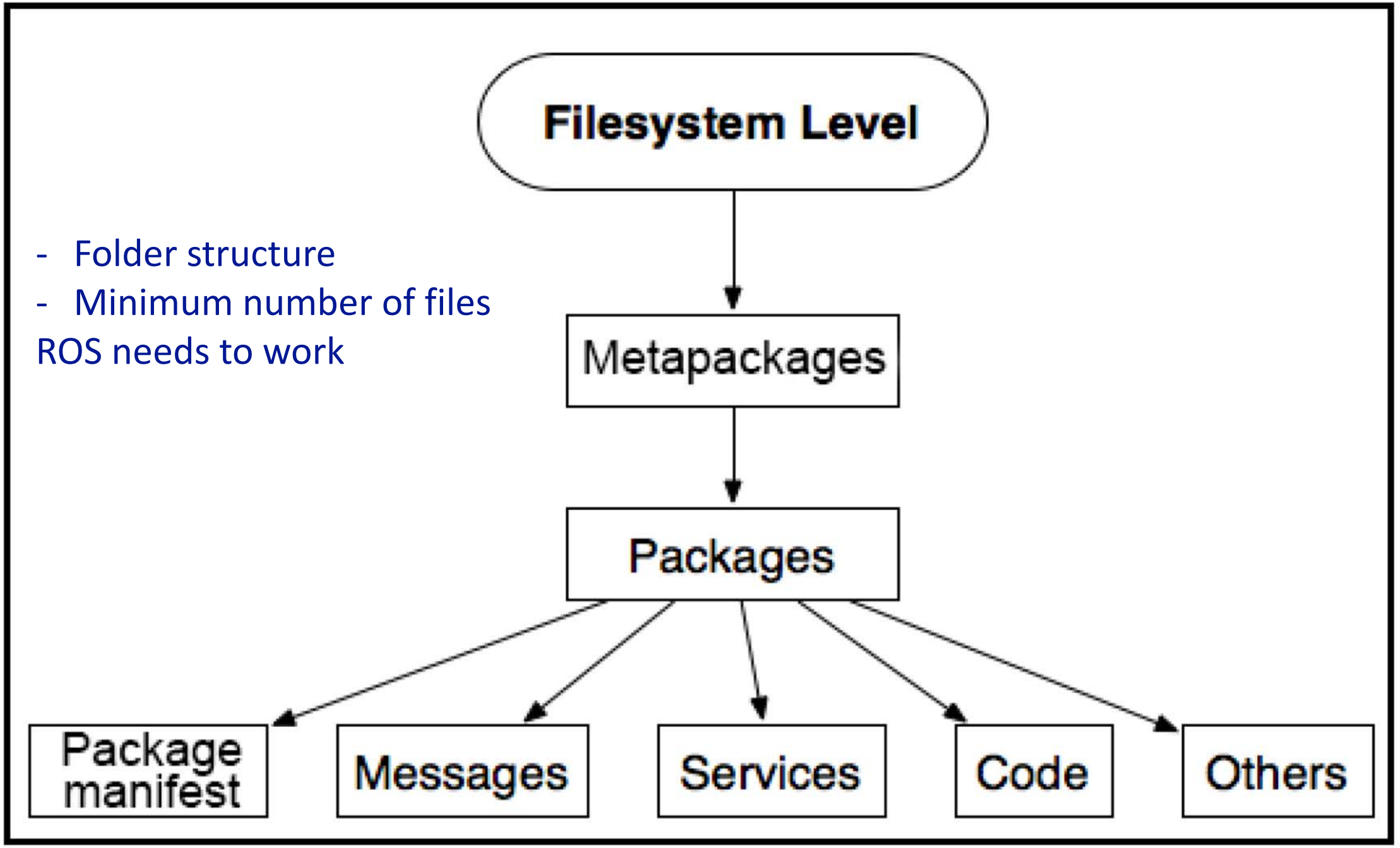


This Lecture

- Build System in ROS
- ROS Workspace
- Packages
- package.xml
- CMakeLists.txt

- Folder structure
- Minimum number of files

ROS needs to work



The main goal of the ROS Filesystem is to centralize the build process of a project, while at the same time provide enough flexibility and tooling to decentralize its dependencies.

catkin Build System

- **catkin** is the ROS build system
  - The set of tools that ROS uses to generate executable programs, libraries and interfaces
- The original ROS build system was **rosbuild**
  - Still used for older packages (before 2015)
- Implemented as custom CMake macros along with some Python code

Just FYI: CMake is an open-source, cross-platform family of tools designed to build, test and package software.

rosbuild

- The former ROS build system
  - But still used by many packages
- Collection of custom scripts + CMake + make
- Suffered from some limitations
  - Mostly impacting ROS packagers
    - Those tasked with turning ROS source code into redistributable binaries
    - Portability, standards compliance, cross-compiling, ease of packaging
  - Worked quite well for researchers / rapid prototyping
- Should not be used for new ROS modules
- http://www.ros.org/wiki/rosbuild

catkin

- The current ROS build system
  - Official as of ROS Groovy
- Goals:
  - Reuse existing tools (CMake macros)
  - Be as standard compliant as possible
  - Facilitate binary packaging
    - “install” target
- Modules using catkin are called “wet”
  - Packages still using rosbuild are called “dry”
  - Analogy of a rising water line
    - rosbuild packages can depend on catkin packages, but not the other way around
    - ROS modules are converted to catkin from the bottom of the dependency chain to the top

Catkin: Slim, cylindrical flower cluster from a willow tree

Why catkin?

Limitation of rosbuild

- *make invoking CMake invoking make?!*

  - why does "rosmake MyPkg" take so long...
- *difficult to cross-compile*
- *no "install" target*

  - everything (except . and *.o files) gets packaged in debs*
- *distribution of packages not compliant to Filesystem Hierarchy Standard (FHS)*

catkin = cmake + X

catkin is based on CMake

- *catkin utilizes packaging conventions like*
  - *find_package() infrastructure*
  - *pkg-config*
- *extends CMake with some "nice to have" features like*
  - *enables to use packages after building (without installation)*
  - *generates find_package() code for your package*
  - *generates pkg-config files for your package*
  - *handles build order of multiple packages*
  - *handles transitive dependencies*
  - *(optionally) builds multiple packages with a single CMake invocation (for performance)*

CMake

Is a widely used standard

- *common CMake workflow*
  - *mkdir build && cd build*
  - *cmake ..*
  - *make*

Enables a lot of features

- *e.g. cross-compilation*

But is also very explicit and requires "more" stuff in CMakeLists.txt than rosbuild

- *e.g. explicitly installing artifacts*

Workflow

With catkin and bloom from source to (Debian) packages

catkin - build system

- *makes it more efficient to build your package(s)*
- *makes your package more standard compliant, hence reusable by others*

bloom - release tool

- *supports you in creating (Debian) packages*
- *makes applying patches during the process easy*

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 Workspace

- A set of directories in which a set of related ROS code lives
- You can have multiple ROS workspaces, but you can only work in one of them at any one time
- Contains the following spaces:

Source space	Contains the source code of catkin packages. Each folder within the source space contains one or more catkin packages.
Build Space	is where CMake is invoked to build the catkin packages in the source space. CMake and catkin keep their cache information and other intermediate files here.
Development (Devel) Space	is where built targets are placed prior to being installed
Install Space	Once targets are built, they can be installed into the install space by invoking the install target.

Packages organization in the ROS 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
```

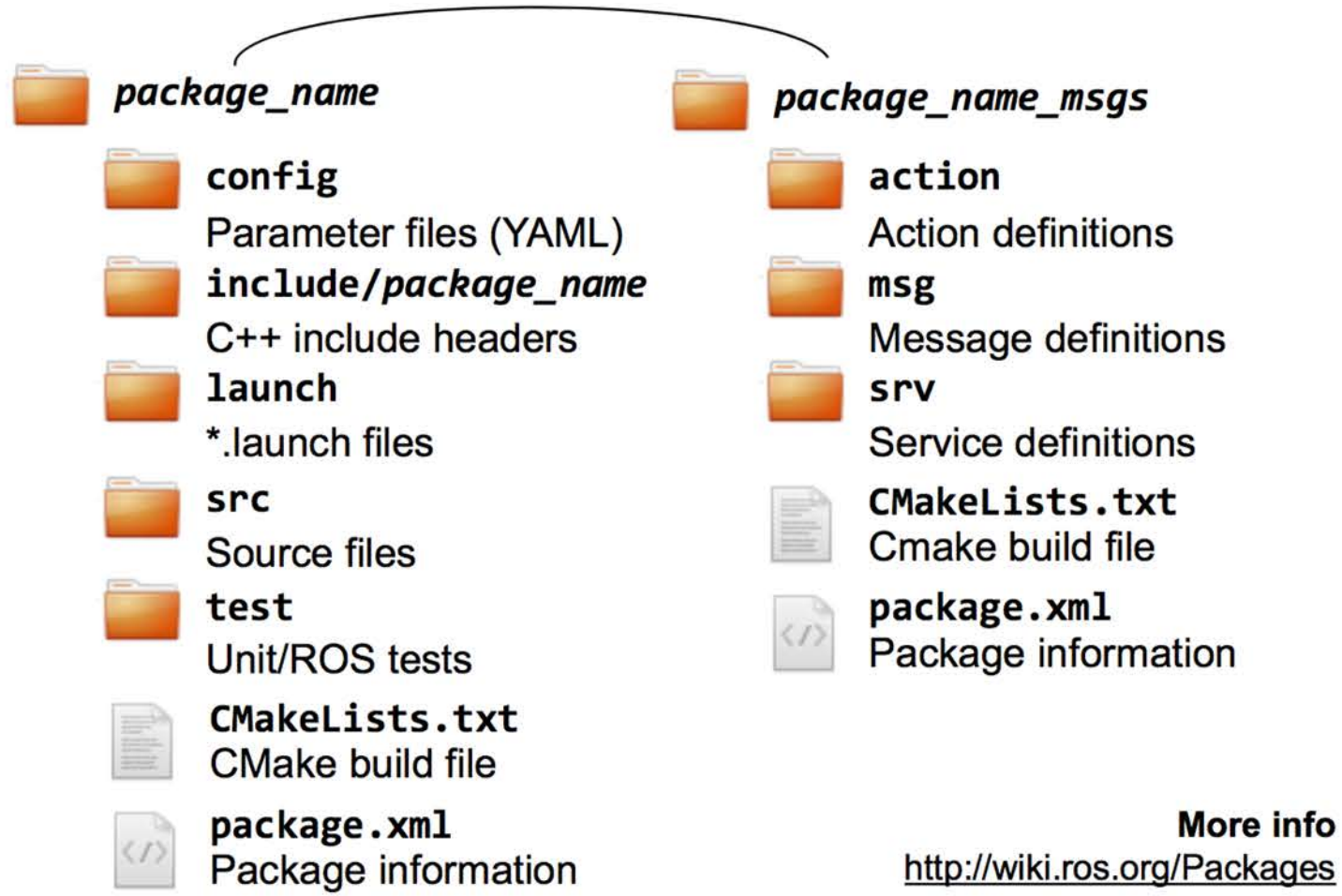
ROS Packages

- 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
http://wiki.ros.org/Packages

```
workspace_folder/         -- WORKSPACE
  src/                    -- SOURCE SPACE
    CMakeLists.txt        -- The 'toplevel' CMake file
    package_1/
      CMakeLists.txt
      package.xml
      ...
    package_n/
      CATKIN_IGNORE       -- Optional empty file to exclude package_n from being processed
      CMakeLists.txt
      package.xml
      ...
  build/                  -- BUILD SPACE
    CATKIN_IGNORE         -- Keeps catkin from walking this directory
  devel/                  -- DEVELOPMENT SPACE (set by CATKIN_DEVEL_PREFIX)
    bin/
    etc/
    include/
    lib/
    share/
    .catkin
    env.bash
    setup.bash
    setup.sh
    ...
  install/                -- INSTALL SPACE (set by CMAKE_INSTALL_PREFIX)
    bin/
    etc/
    include/
    lib/
    share/
    .catkin
    env.bash
    setup.bash
    setup.sh
    ...
```

Example Workspace Structure

```
~/my_workspace/
├── build/
├── devel/
└── src/
    ├── CMakeLists.txt -> /opt/ros/$ROS_DISTRO/share/catkin/cmake/toplevel.cmake
    ├── your_awesome_package/
    ├── my_awesome_package/
    └── his_terrible_package/
```

Creating a catkin Workspace

- Initially, the workspace will contain only the top-level CMakeLists.txt

```
$ mkdir ~/catkin_ws/src
$ cd ~/catkin_ws/src
$ catkin_init_workspace
```

- **catkin_make** command builds the workspace and all the packages within it

```
cd ~/catkin_ws
catkin_make
```

Sourcing the Workspace

If you want to interact with your workspace like you're used to, you need to source the `setup.sh` file that catkin generates for you. This will add all the appropriate environment variables that would be otherwise too complicated to manage yourself:

```
source catkin_workspace/devel/setup.sh
```

Recall: Sourcing the Workspace

It's convenient if the ROS environment variables are automatically added to your bash session every time a new shell is launched:

```
echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

How to build multiple packages? Workspaces

Arrange them in a workspace

/ws/# root of the workspace

/ws/src/# source space

/ws/src/repoA

/ws/src/repoB

/ws/src/pkgC

/ws/src/pkgD

How to build multiple packages? catkin_make

- *Sourcing your ROS environment, commonly*

  source /opt/ros/kinetic/setup.bash

- *Invoking catkin_make in the root of the workspace will create the subfolders*

  /ws/build/ # build space

  /ws/devel/ # devel space (FHS layout)

- *Invoking catkin_make install installs the packages to*

  /ws/install/ # install space

Overview of a catkin package

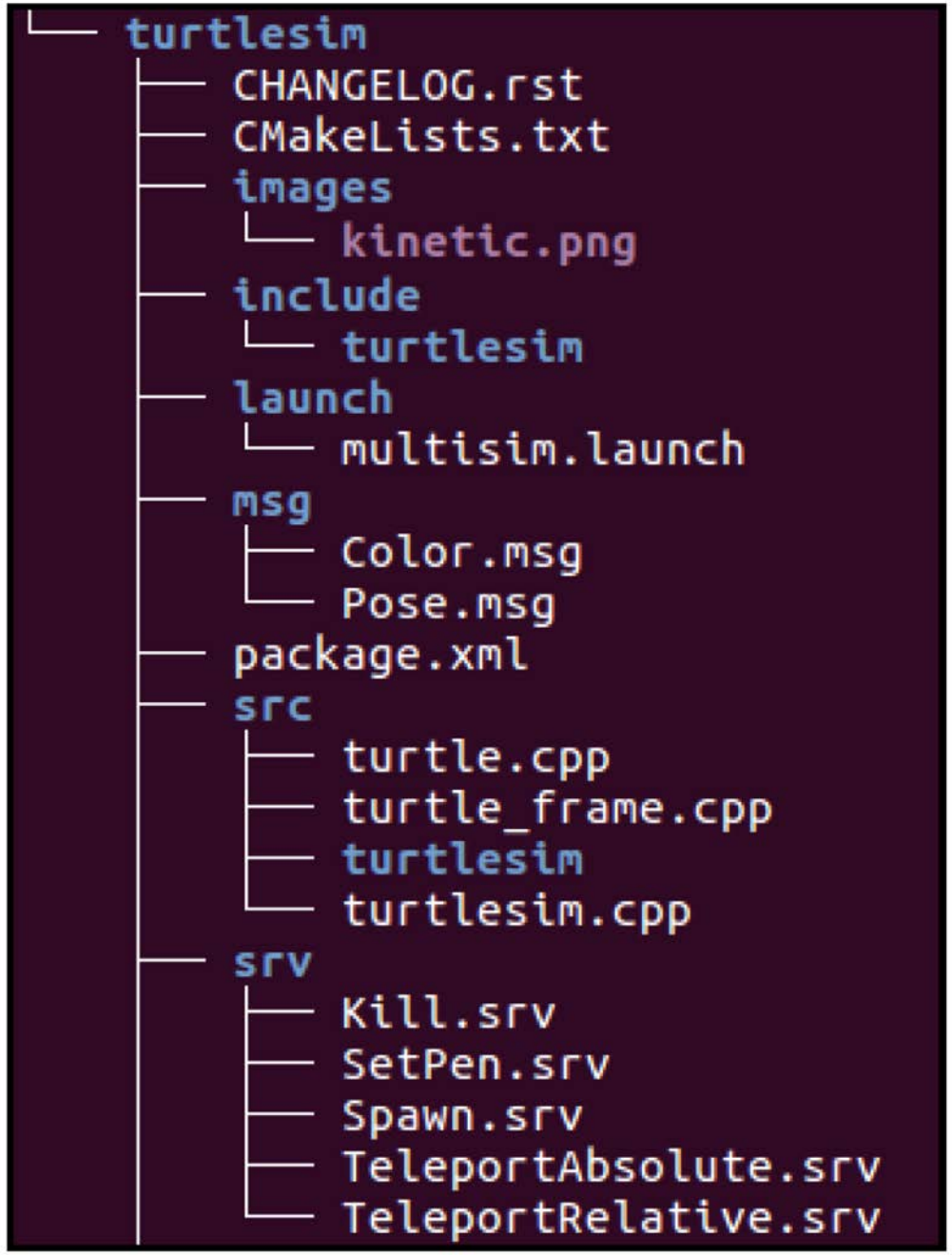
package.xml *(as specified in REP 127)*

- *Contains the meta information of a package*
  - *name, description, version, maintainer(s), license*
  - *opt. authors, url's, dependencies, plugins, etc...*

CMakeLists.txt

- *The main CMake file to build the package*
- *Calls catkin-specific functions/macros*
  - *"Read" the package.xml*
  - *find other catkin packages to access libraries / include directories*
  - *export items for other packages depending on you*

Turtlesim Package



ROS Packages
package.xml

Must be included with any catkin-compliant package's root folder.

- 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 template for ROS packages.</description>
  <maintainer email="pfankhauser@e…">Peter Fankhauser</maintainer>
  <license>BSD</license>
  <url type="website">https://github.com/ethz-asl/ros_best_pr…</url>
  <author email="pfankhauser@ethz.ch">Peter Fankhauser</author>

  <buildtool_depend>catkin</buildtool_depend>

  <depend>roscpp</depend>
  <depend>sensor_msgs</depend>
</package>
```

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

ROS Packages
CMakeLists. txt

The `CMakeLists.txt` is the input to the CMakebuild system

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

CMakeLists.txt

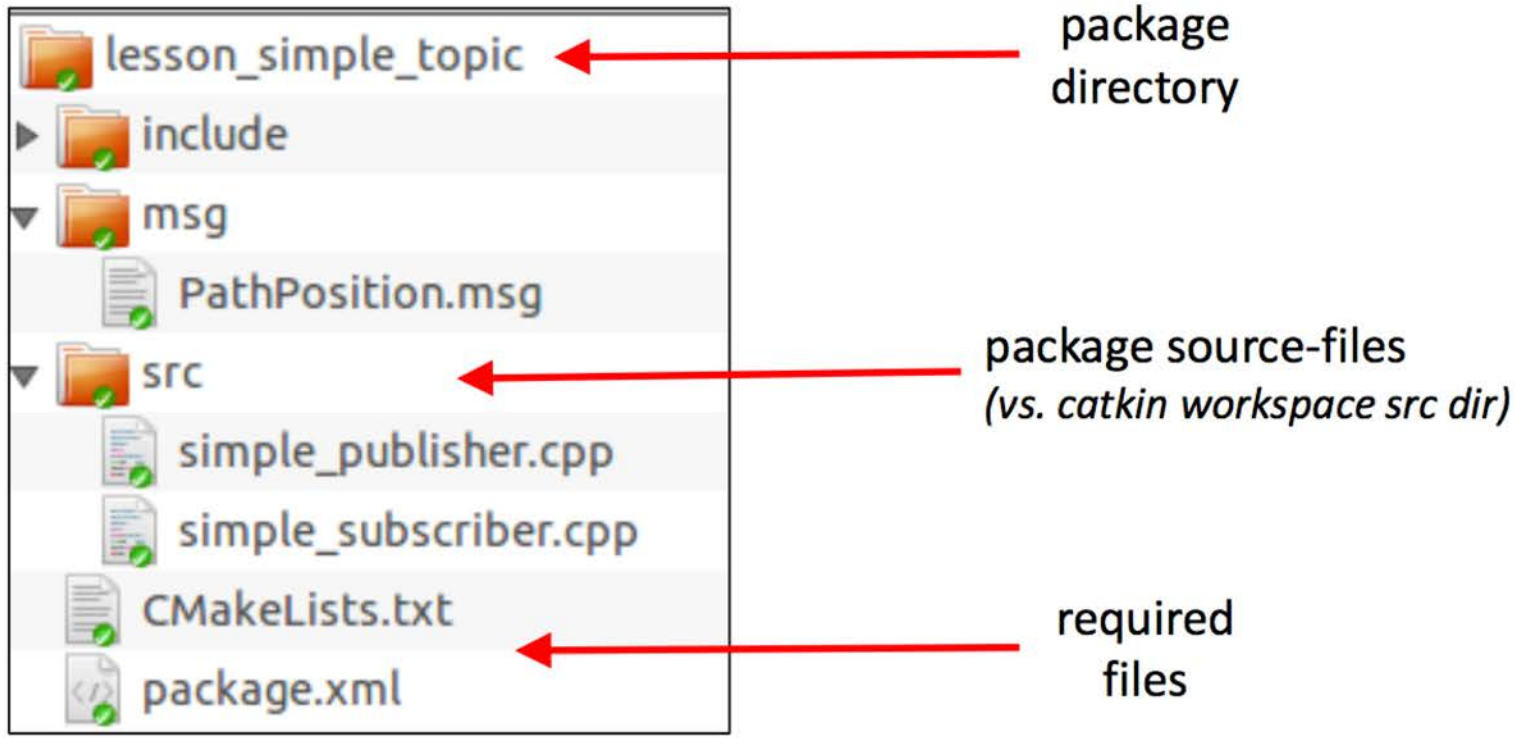
```
cmake_minimum_required(VERSION 2.8.3)
project(ros_package_template)

## Use C++11
add_definitions(--std=c++11)

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

…
```

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

Common Files and Directories

Directory	Explanation
include/	C++ include headers
src/	Source files
msg/	Folder containing Message (msg) types
srv/	Folder containing Service (srv) types
launch/	Folder containing launch files
package.xml	The package manifest
CMakeLists.txt	CMake build file

- ROS components are organized into **packages**
- Packages contain several **required files**:
  - `package.xml`
    - **metadata** for ROS: package name, description, dependencies, ...
  - `CMakeLists.txt`
    - **build rules** for catkin



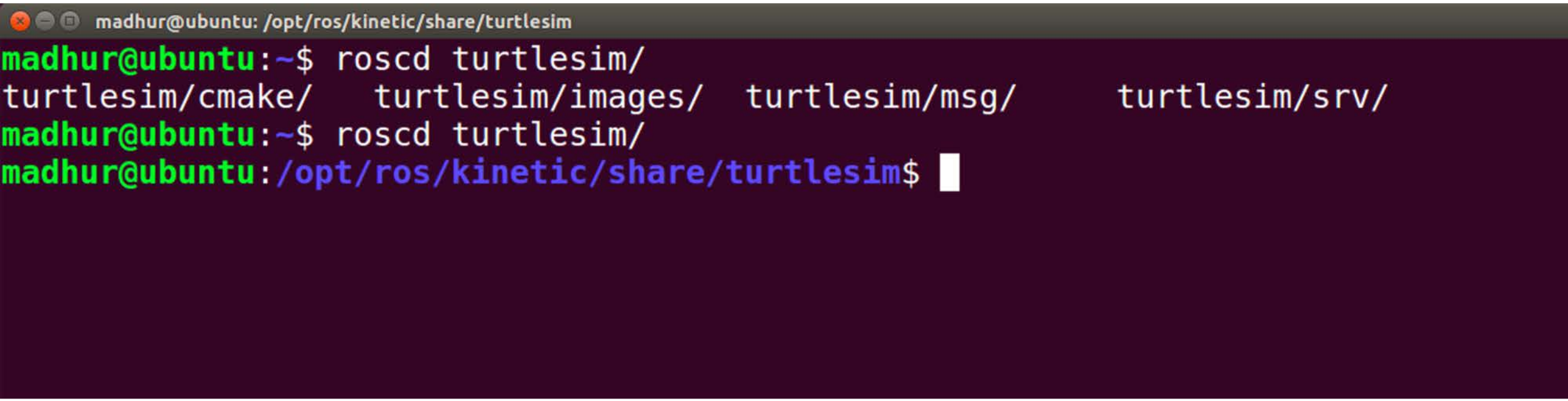
Where are the Packages?

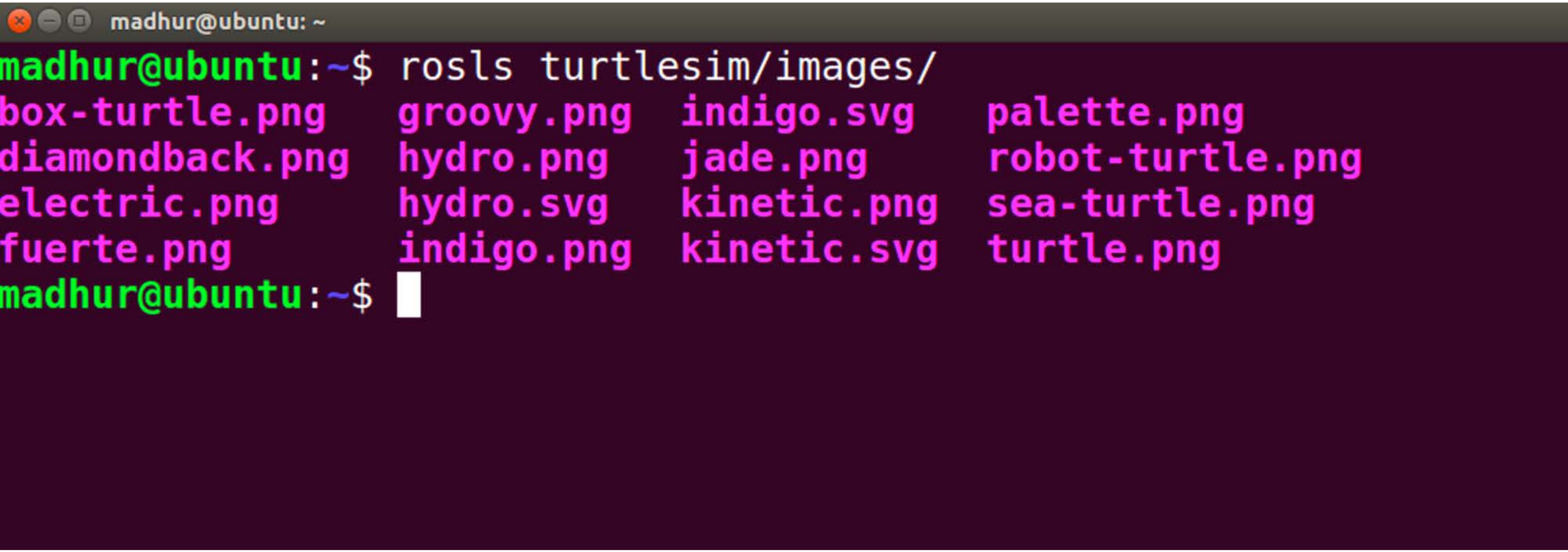
- **Installed packages: *`/opt/ros/indigo`***
  - **Package parts divided into *lib*, *include*, and *share* subdirectories**
    - **Filesystem Hierarchy Standard (FHS)**
  - **Owned by root**
    - **Look, but don't touch**
  - **Typically does not include the source code (except for Python)**
- **User packages: anywhere**
  - **But usually in user's home directory**

Anywhere? How does ROS find the packages?

Navigating Across ROS Packages

- Users typically have many ROS packages in many places
- Frequently need to:
  - Jump from one package to another
  - Reference a file in another package
- rosbash suite – command line tools for quick package navigation & use
  - `roscd` - change directory to a package name
    - Example: `roscd robot_spinner/src`
  - `rospd` - pushd equivalent of roscd
  - `rosd` - lists directories in the directory-stack
  - `rosls` - list files of a ros package
  - `rosed` - edit a file in a package
  - `roscp` - copy a file from a package
  - `rosrun` - run executables of a ros package
- `rospack` – get detailed information about a a package
  - Examples:
    ```
    rospack find robot_spinner
    rospack depends robot_spinner
    ```
  - Replicate roscd functionality: `cd `rospack find robot_spinner`/src`


madhur@ubuntu: /opt/ros/kinetic/share/turtlesim
madhur@ubuntu:~$ roscd turtlesim/
turtlesim/cmake/   turtlesim/images/ turtlesim/msg/   turtlesim/srv/
madhur@ubuntu:~$ roscd turtlesim/
madhur@ubuntu:/opt/ros/kinetic/share/turtlesim$


madhur@ubuntu: ~
madhur@ubuntu:~$ rosls turtlesim/images/
box-turtle.png groovy.png indigo.svg palette.png
diamondback.png hydro.png jade.png robot-turtle.png
electric.png hydro.svg kinetic.png sea-turtle.png
fuerte.png indigo.png kinetic.svg turtle.png
madhur@ubuntu:~$

package.xml – package meta information supporting documentation, compilation, and distribution

- Required: name, description, version, maintainer(s), license
- Optional: authors, url's, dependencies, plugins, …

CMakeLists.txt – the main CMake file used to build the package

- Standard and catkin-specific CMake functions / macros
  - "Reads" the package.xml file
  - Finds other catkin packages to access libraries / include directories
  - Export items for other packages depending on you

ROS packages

- To create a new package, navigate to your workspace and then use the **catkin_create_pkg** utility.

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg beginner_tutorials std_msgs
rospy roscpp
```

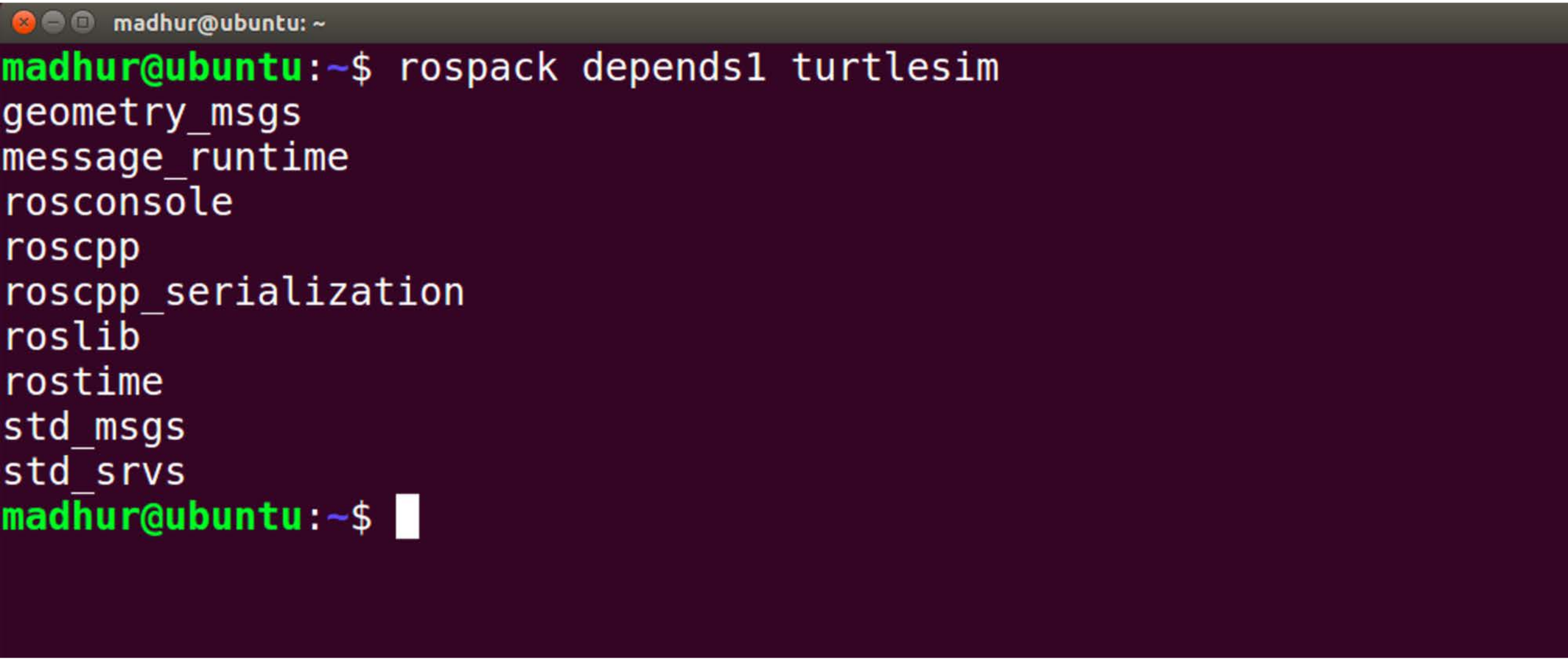
- To build the new created package do:

```
$ cd ~/catkin_ws
$ catkin_make
$ . ~/catkin_ws/devel/setup.bash
$ rospack profile
```

Check package dependencies

```
$ rospack depends1 beginner_tutorials
```

```
roscpp
rospy
std_msgs
```


madhur@ubuntu: ~
madhur@ubuntu:~$ rospack depends1 turtlesim
geometry_msgs
message_runtime
rosconsole
roscpp
roscpp_serialization
roslib
rostime
std_msgs
std_srvs
madhur@ubuntu:~$

What goes in the package.xml? Required tags

```
<?xml version="1.0"?>
<package>
  <name>rospy_tutorials</name>
  <version>0.3.12</version>
  <description>
    This package attempts to show the features of ROS Python API
    step-by-step, including using messages, servers, parameters, etc.
    These tutorials are compatible with the nodes in roscpp_tutorial.
  </description>
  <maintainer email="dthomas@osrfoundation.org">
    Dirk Thomas
  </maintainer>
  <license>BSD</license>
  ...
</package>
```

What goes in the package.xml? Different dependencies

<build_depend>

dependencies to build your package

<buildtool_depend>

tools used on the building platform
to build your package, usually catkin

<run_depend>

dependencies other packages need to build
against your package or use your package
- if the dependency provides a shared library
- if the headers of the dependencies are exposed

<test_depend>

additional dependencies to run tests

package.xml : Description tag

First update the description tag:

Toggle line numbers

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

Change the description to anything you like, but by convention the first sentence should be short while covering the scope of the package. If it is hard to describe the package in a single sentence then it might need to be broken up.

package.xml : Maintainer tag

Next comes the maintainer tag:

Toggle line numbers

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

This is a required and important tag for the package.xml because it lets others know who to contact about the package. At least one maintainer is required, but you can have many if you like. The name of the maintainer goes into the body of the tag, but there is also an email attribute that should be filled out:

Toggle line numbers

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

package.xml : license tag

Next is the license tag, which is also required:

Toggle line numbers

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

You should choose a license and fill it in here. Some common open source licenses are BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, and LGPLv3. You can read about several of these at the Open Source Initiative. For this tutorial we'll use the BSD license because the rest of the core ROS components use it already:

Toggle line numbers

```
8   <license>BSD</license>
```

package.xml : Build dependencies

Toggle line numbers

```
27   <!-- The *_depend tags are used to specify dependencies -->
28   <!-- Dependencies can be catkin packages or system dependencies -->
29   <!-- Examples: -->
30   <!-- Use build_depend for packages you need at compile time: -->
31   <!--   <build_depend>genmsg</build_depend> -->
32   <!-- Use buildtool_depend for build tool packages: -->
33   <!--   <buildtool_depend>catkin</buildtool_depend> -->
34   <!-- Use exec_depend for packages you need at runtime: -->
35   <!--   <exec_depend>python-yaml</exec_depend> -->
36   <!-- Use test_depend for packages you need only for testing: -->
37   <!--   <test_depend>gtest</test_depend> -->
38   <buildtool_depend>catkin</buildtool_depend>
39   <build_depend>roscpp</build_depend>
40   <build_depend>rospy</build_depend>
41   <build_depend>std_msgs</build_depend>
```

package.xml : Execution dependencies

Toggle line numbers

```
12    <buildtool_depend>catkin</buildtool_depend>
13
14    <build_depend>roscpp</build_depend>
15    <build_depend>rospy</build_depend>
16    <build_depend>std_msgs</build_depend>
17
18    <exec_depend>roscpp</exec_depend>
19    <exec_depend>rospy</exec_depend>
20    <exec_depend>std_msgs</exec_depend>
```

Toggle line numbers

```xml
<?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>
```

package.xml Example

```
<?xml version="1.0"?>
<package>
    <name>robot_spinner</name>
    <version>0.2.12</version>
    <description>Motion planner to spin a robot around</description>
    <maintainer email="boulet@ll.mit.edu">Michael Boulet</maintainer>
    <license>BSD</license>

    <url type="website">http://www.ros.org/wiki/robot_spinner</url>
    <author>Michael Boulet</author>

    <buildtool_depend>catkin</buildtool_depend>

    <build_depend>roscpp</build_depend>
    <build_depend>geometry_msgs</build_depend>

    <run_depend>roscpp</run_depend>
    <run_depend>geometry_msgs</run_depend>

    <export>
        <nodelet plugin="${prefix}/robot_spinner_nodelet.xml"/>
    </export>
</package>
```

Should match the directory name

Tools used on building platform needed to build this package

Dependencies needed to build this package

Dependencies needed to run this package / dependencies other packages need to build against or use this package (often the same as `build_depend`)

Container for "extra information" subsystems or other packages need to embed

What goes in the CMakeLists.txt?

```
cmake_minimum_required(VERSION 2.8.3)
project(roscpp_tutorials)  # same package name as in package.xml

find_package(catkin REQUIRED COMPONENTS
    message_generation rosconsole roscpp roscpp_serialization rostime)
find_package(Boost REQUIRED COMPONENTS date_time thread)

include_directories(  # consider reasonable include order
    include ${catkin_INCLUDE_DIRS} ${Boost_INCLUDE_DIRS})

add_library(mylib src/file1.cpp src/file2.cpp)  # same for add_executable
target_link_libraries(mylib ${catkin_LIBRARIES} ${Boost_LIBRARIES})

catkin_package(  # information exposed to other packages
    INCLUDE_DIRS include
    LIBRARIES mylib
    CATKIN_DEPENDS message_runtime std_msgs)
```

What goes in the CMakeLists.txt? Installing all artifacts

```
# executables are installed to lib/PKGNAME to be rosrun-able
install(TARGETS myexe
    RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION})

# scripts are installed under lib/PKGNAME to be rosrun-able
# only a selected set of core binaries should go into the global bin
install(PROGRAMS myscript
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION})

# libraries are installed to lib (or bin for dll's under Windows)
install(TARGETS mylib
    RUNTIME  DESTINATION  ${CATKIN_GLOBAL_BIN_DESTINATION}
    ARCHIVE  DESTINATION  ${CATKIN_PACKAGE_LIB_DESTINATION}
    LIBRARY DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION})
```

What goes in the CMakeLists.txt? Installing all artifacts

```
# header are installed to include (within their namespaced subfolder)
install(DIRECTORY include/PKGNAME/   # note the trailing slash
    DESTINATION ${CATKIN_PACKAGE_INCLUDE_DESTINATION}
    FILES_MATCHING PATTERN "*.h")   # skip hidden files/folders

# other files/folders are installed to share/PKGNAME
install(FILES otherfile.txt
    DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION})
install(DIRECTORY myfolder   # note the missing trailing slash
    DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION})
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8.3)
project(robot_spinner)

## Find catkin macros and libraries
## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
## is used, also find other catkin packages
find_package(catkin REQUIRED COMPONENTS roscpp std_msgs)

## System dependencies are found with CMake's conventions
# find_package(Boost REQUIRED COMPONENTS system)


## Uncomment this if the package has a setup.py. This macro ensures
## modules and global scripts declared therein get installed
## See http://ros.org/doc/api/catkin/html/user_guide/setup_dot_py.html
# catkin_python_setup()

######################################
## Declare ROS messages and services ##
######################################

## Generate messages in the 'msg' folder
# add_message_files(
#   FILES
#   Message1.msg
#   Message2.msg
# )

## Generate services in the 'srv' folder
# add_service_files(
#   FILES
#   Service1.srv
#   Service2.srv
# )

## Generate added messages and services with any dependencies listed here
# generate_messages(
#   DEPENDENCIES
#   std_msgs
# )

###################################
## catkin specific configuration ##
###################################
## The catkin_package macro generates cmake config files for your package
## Declare things to be passed to dependent projects
## INCLUDE_DIRS: uncomment this if you package contains header files
## LIBRARIES: libraries you create in this project that dependent projects also need
## CATKIN_DEPENDS: catkin_packages dependent projects also need
## DEPENDS: system dependencies of this project that dependent projects also need
catkin_package(
#  INCLUDE_DIRS include
#  LIBRARIES robot_spinner
#  CATKIN_DEPENDS roscpp std_msgs
#  DEPENDS system_lib
)

###########
## Build ##
###########

## Specify additional locations of header files
## Your package locations should be listed before other locations
# include_directories(include)
include_directories(
  ${catkin_INCLUDE_DIRS}
)

## Declare a cpp library
# add_library(robot_spinner
#   src/${PROJECT_NAME}/robot_spinner.cpp
# )

## Declare a cpp executable
# add_executable(robot_spinner_node src/robot_spinner_node.cpp)

## Add cmake target dependencies of the executable/library
## as an example, message headers may need to be generated before nodes
# add_dependencies(robot_spinner_node robot_spinner_generate_messages_cpp)

## Specify libraries to link a library or executable target against
# target_link_libraries(robot_spinner_node
#   ${catkin_LIBRARIES}
# )

#############
## Install ##
#############

# all install targets should use catkin DESTINATION variables
# See http://ros.org/doc/api/catkin/html/adv_user_guide/variables.html

## Mark executable scripts (Python etc.) for installation
## in contrast to setup.py, you can choose the destination
# install(PROGRAMS
#   scripts/my_python_script
#   DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
# )

## Mark executables and/or libraries for installation
# install(TARGETS robot_spinner robot_spinner_node
#   ARCHIVE DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
#   LIBRARY DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
#   RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
# )

## Mark cpp header files for installation
# install(DIRECTORY include/${PROJECT_NAME}/
#   DESTINATION ${CATKIN_PACKAGE_INCLUDE_DESTINATION}
#   FILES_MATCHING PATTERN "*.h"
#   PATTERN ".svn" EXCLUDE
# )

## Mark other files for installation (e.g. launch and bag files, etc.)
# install(FILES
#   # myfile1
#   # myfile2
#   DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION}
# )

#############
## Testing ##
#############

## Add gtest based cpp test target and link libraries
# catkin_add_gtest(${PROJECT_NAME}-test test/test_robot_spinner.cpp)
# if(TARGET ${PROJECT_NAME}-test)
#   target_link_libraries(${PROJECT_NAME}-test ${PROJECT_NAME})
# endif()

## Add folders to be run by python nosetests
# catkin_add_nosetests(test)
```

- **Complex and non-intuitive**
- **Template generated by `catkin_create_pkg` is very descriptive**
  - **Very long, includes most features of catkin**
  - **Typical CMakeLists.txt files only use a few features**
- **Typical workflows:**
  - **Uncomment and edit the relevant lines in the `catkin_create_pkg` template**
  - **Find a similar package and copy / edit their CMakeLists.txt file for your program**

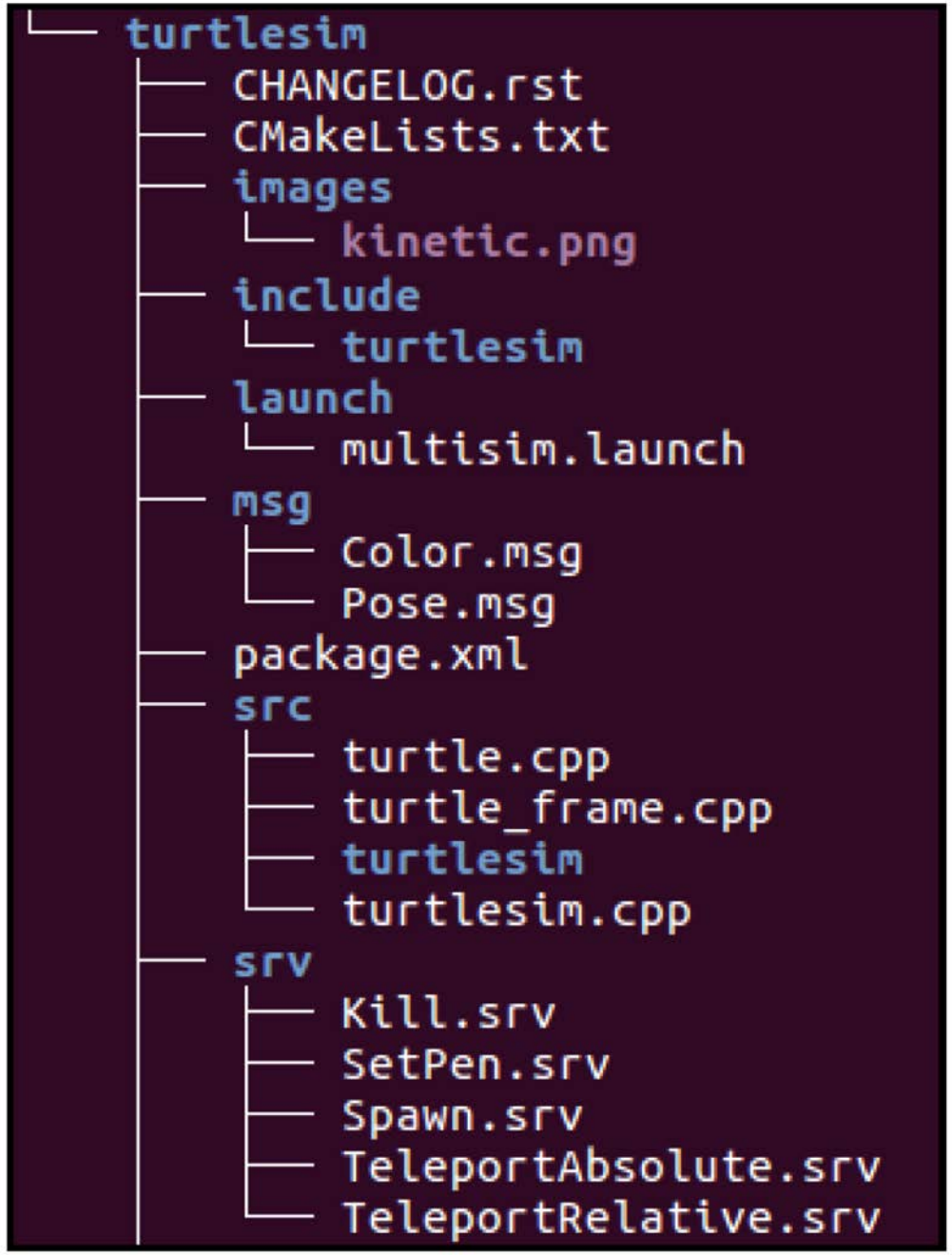
CMakeLists.txt Example (Python)

```
cmake_minimum_required(VERSION 2.8.3)
project(rospy_tutorials)

find_package(catkin REQUIRED COMPONENTS message_generation rostest
std_msgs)

add_message_files(DIRECTORY msg FILES Floats.msg HeaderString.msg)
add_service_files(DIRECTORY srv FILES AddTwoInts.srv BadTwoInts.srv)

generate_messages(DEPENDENCIES std_msgs)

catkin_package(CATKIN_DEPENDS message_runtime std_msgs)

add_rostest(test/test-add-two-ints.launch)

install(PROGRAMS
  005_add_two_ints/add_two_ints_client.py
  005_add_two_ints/add_two_ints_server.py
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION} )
```

CMakeLists.txt Example (C++)

```
cmake_minimum_required(VERSION 2.8.3)
project(roscpp_tutorials)

find_package(Boost REQUIRED COMPONENTS date_time thread)
find_package(catkin REQUIRED COMPONENTS message_generation rostime
roscpp rosconsole roscpp_serialization)

include_directories(${catkin_INCLUDE_DIRS})
link_directories(${catkin_LIBRARY_DIRS})

add_service_files(DIRECTORY srv FILES TwoInts.srv)
generate_messages(DEPENDENCIES std_msgs)

catkin_package(CATKIN_DEPENDS message_runtime std_msgs)

add_executable(babbler src/babbler.cpp)
target_link_libraries(babbler ${catkin_LIBRARIES} ${Boost_LIBRARIES})
add_dependencies(babbler roscpp_tutorials_gencpp)
install(TARGETS babbler
    RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION})
```

Turtlesim Package



Turtlesim Package

Branch: **melodic-devel** ▾ **ros_tutorials** / **turtlesim** /

dirk-thomas add shortcut to quit teleop (#59)

..	
images	Add melodic turtle. (#41)
include/turtlesim	add disable code to spawn all available turtle types
launch	changing turtlesim to turtlesim_node for tutorial clarity
msg	remove turtlesim velocity and use Twist msg
src	sin(x+pi/2)-> cos(x); cos(x+pi/2)->-sin(x) in turtle.cpp. Fixing #47 (#...
srv	Optionally name your turtles yourself
tutorials	add shortcut to quit teleop (#59)
CHANGELOG.rst	0.9.1
CMakeLists.txt	update to Qt5
package.xml	0.9.1

ros / ros_tutorials

Code | Issues 3 | Pull requests 2 | Actions | Security | Insights

Branch: melodic-devel ros_tutorials / turtlesim / src /

zhoulaifu and dirk-thomas sin(x+pi/2)-> cos(x); cos(x+pi/2)->-sin(x) in turtle.cpp. Fixing #47 (#...

..

turtlesim	Add turtlesim to the ros_tutorials stack
turtle.cpp	sin(x+pi/2)-> cos(x); cos(x+pi/2)->-sin(x) in turtle.cpp. Fixing #47 (#...
turtle_frame.cpp	Add melodic turtle. (#41)
turtlesim.cpp	TurtleApp accepts argc by reference

ros / ros_tutorials

Code | Issues 3 | Pull requests 2 | Actions | Security | Insights

Branch: melodic-devel ros_tutorials / turtlesim / tutorials /

dirk-thomas add shortcut to quit teleop (#59)

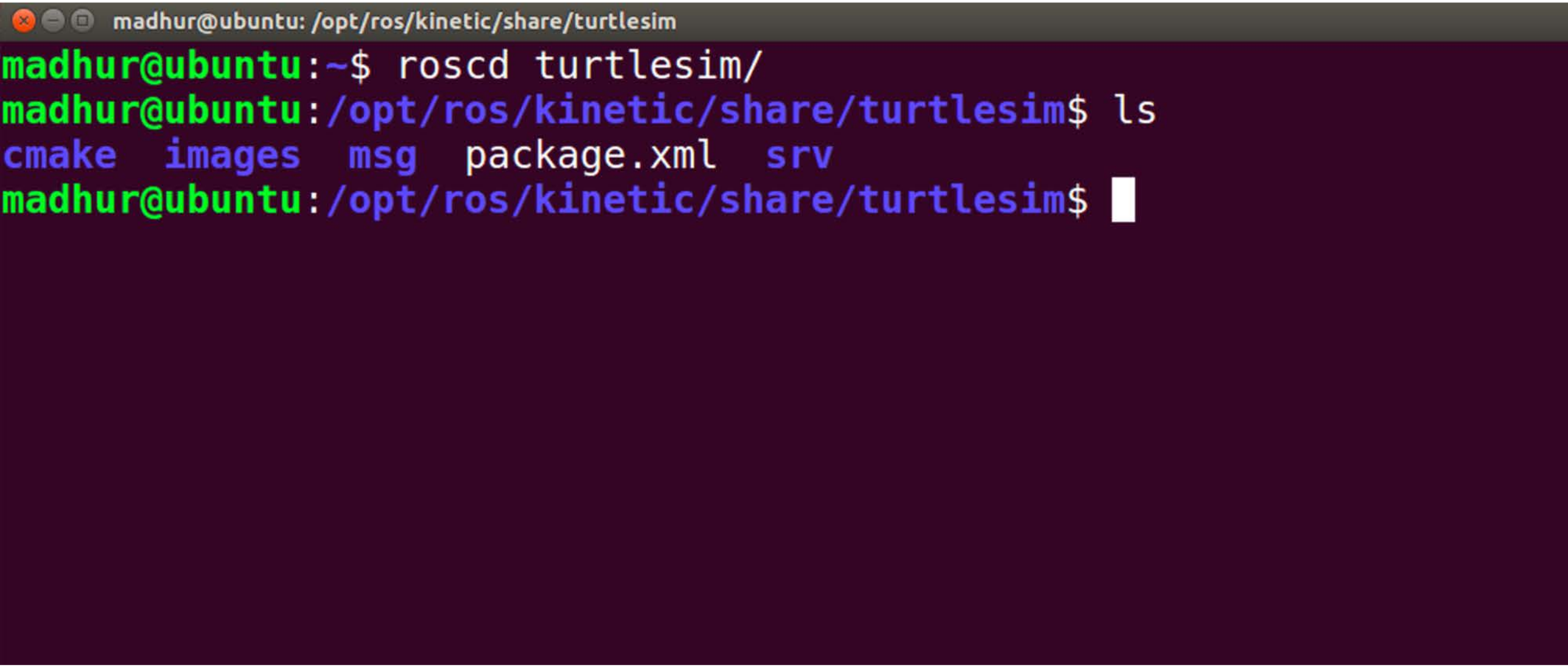
..	
draw_square.cpp	Fixing issue #43, with [-pi, pi) scope for theta (#46)
mimic.cpp	chaning command_velocity to cmd_vel
teleop_turtle_key.cpp	add shortcut to quit teleop (#59)

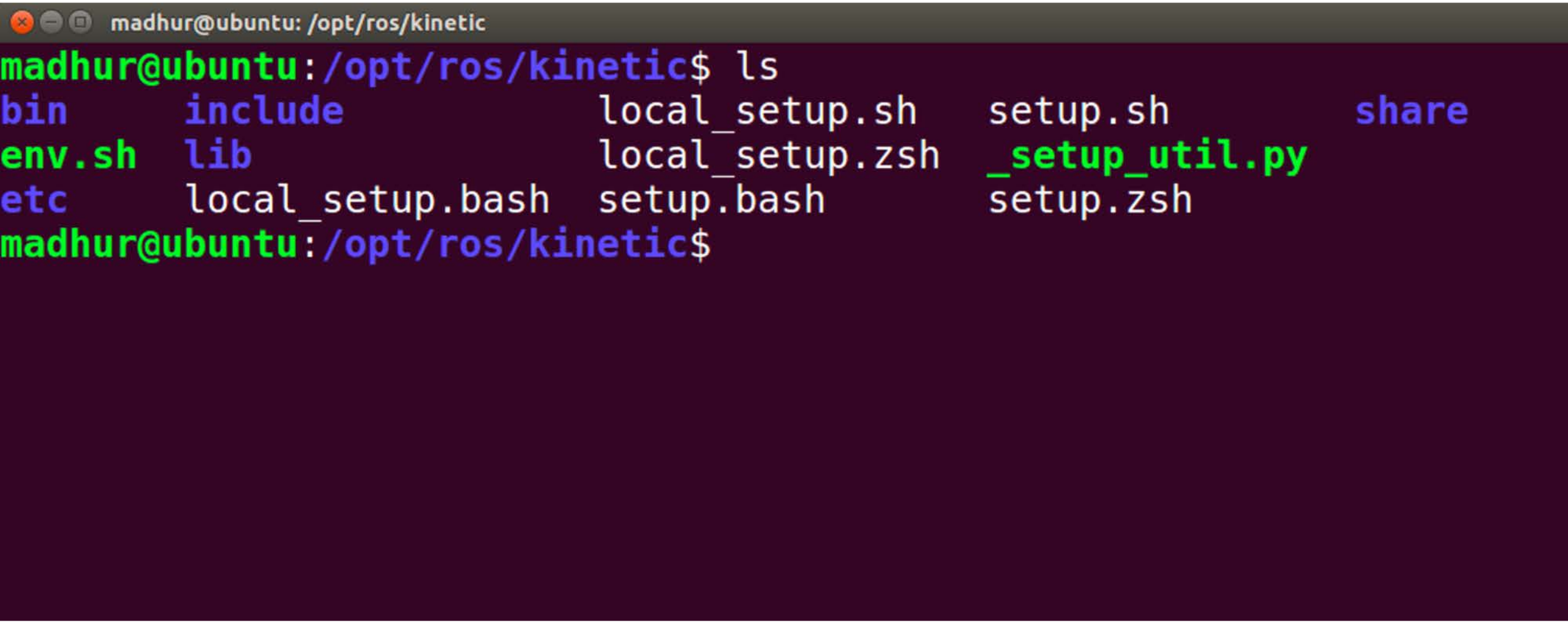
41 lines (37 sloc) | 1.43 KB

Raw Blame History

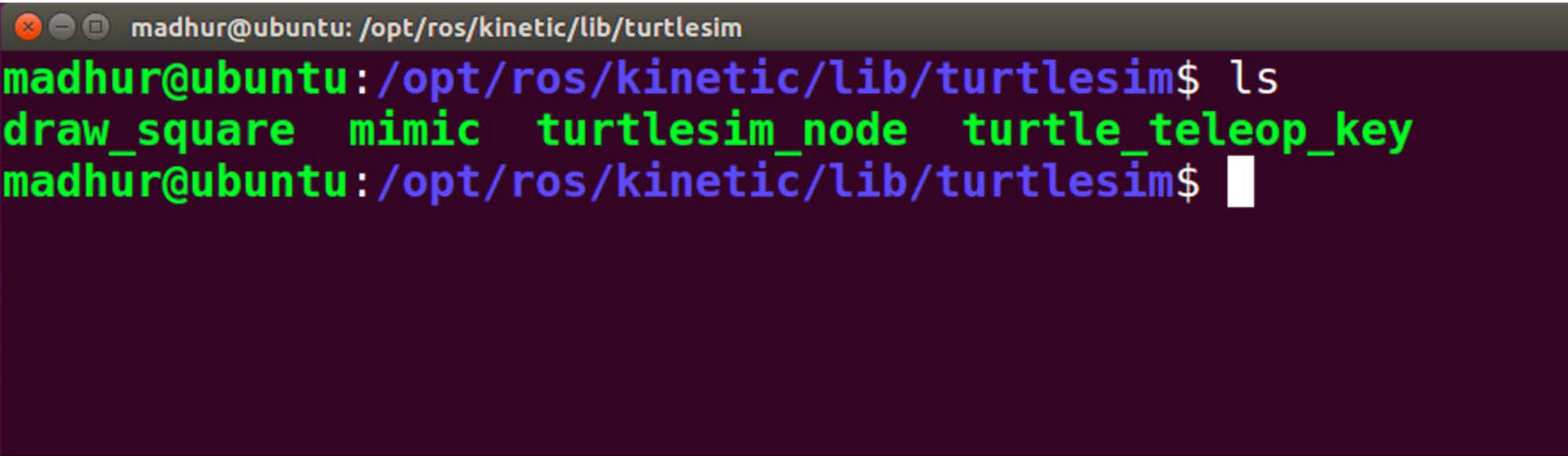
```xml
<?xml version="1.0"?>
<package>
  <name>turtlesim</name>
  <version>0.9.1</version>
  <description>
    turtlesim is a tool made for teaching ROS and ROS packages.
  </description>
  <maintainer email="dthomas@osrfoundation.org">Dirk Thomas</maintainer>
  <license>BSD</license>

  <url type="website">http://www.ros.org/wiki/turtlesim</url>
  <url type="bugtracker">https://github.com/ros/ros_tutorials/issues</url>
  <url type="repository">https://github.com/ros/ros_tutorials</url>
  <author>Josh Faust</author>

  <buildtool_depend>catkin</buildtool_depend>

  <build_depend>geometry_msgs</build_depend>
  <build_depend>qtbase5-dev</build_depend>
  <build_depend>message_generation</build_depend>
  <build_depend>qt5-qmake</build_depend>
  <build_depend>rosconsole</build_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>roscpp_serialization</build_depend>
  <build_depend>roslib</build_depend>
  <build_depend>rostime</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_depend>std_srvs</build_depend>

  <run_depend>geometry_msgs</run_depend>
  <run_depend>libqt5-core</run_depend>
  <run_depend>libqt5-gui</run_depend>
  <run_depend>message_runtime</run_depend>
  <run_depend>rosconsole</run_depend>
  <run_depend>roscpp</run_depend>
  <run_depend>roscpp_serialization</run_depend>
  <run_depend>roslib</run_depend>
  <run_depend>rostime</run_depend>
  <run_depend>std_msgs</run_depend>
```

Turtlesim package.xml


madhur@ubuntu: /opt/ros/kinetic/share/turtlesim
madhur@ubuntu:~$ roscd turtlesim/
madhur@ubuntu:/opt/ros/kinetic/share/turtlesim$ ls
cmake images msg package.xml srv
madhur@ubuntu:/opt/ros/kinetic/share/turtlesim$


madhur@ubuntu: /opt/ros/kinetic
madhur@ubuntu:/opt/ros/kinetic$ ls
bin include local_setup.sh setup.sh share
env.sh lib local_setup.zsh _setup_util.py
etc local_setup.bash setup.bash setup.zsh
madhur@ubuntu:/opt/ros/kinetic$

```
rviz
rviz_plugin_tutorials
rviz_python_tutorial
Stage-4.1
stage_ros
stereo_image_proc
tf
tf2_kdl
tf2_ros
theora_image_transport
topic_tools
turtle_actionlib
turtlesim
turtle_tf
turtle_tf2
visualization_marker_tutorials
x86_64-linux-gnu
xacro
madhur@ubuntu:/opt/ros/kinetic/lib$
```


madhur@ubuntu: /opt/ros/kinetic/lib/turtlesim
madhur@ubuntu:/opt/ros/kinetic/lib/turtlesim$ ls
draw_square mimic turtlesim_node turtle_teleop_key
madhur@ubuntu:/opt/ros/kinetic/lib/turtlesim$

Autoware.AI

The original Autoware project built on ROS 1. Launched as a research and development platform for autonomous driving technology.

Branch: master ▾ **Autoware** / **ros** /

Create new file | Upload files | Find file | History

This branch is 249 commits behind autowarefoundation:master. Pull request Compare

amc-nu Fix license notice in corresponding package.xml — Latest commit `91eb0f8` on Jan 18, 2019

Lets look at the ROS workspace in Autoware

..		
src		
.catkin_workspace		
.gitignore		
catkin_make_release	Use colcon as the build tool (autowarefoundation#1704)	13 months ago
catkin_make_release_cross	Use colcon as the build tool (autowarefoundation#1704)	13 months ago
cleanup	Query password by gksu	4 years ago
colcon_release	[fix] SSD detector, cmake colcon (autowarefoundation#1837)	13 months ago
colcon_release_cross	Use colcon as the build tool (autowarefoundation#1704)	13 months ago
cross_toolchain.cmake	Use colcon as the build tool (autowarefoundation#1704)	13 months ago
cross_toolchain_driveworks.cmake	Feature/gmsl multiple (v2) (autowarefoundation#1683)	14 months ago
dpp	rename dataprocessor to dpp	3 years ago
run	add a newer gnome-terminal support	3 years ago
run_proc_manager	Query password by gksu	4 years ago
shutdown	Remove needless things	3 years ago

Branch: master ▾ **Autoware** / ros / **src** /

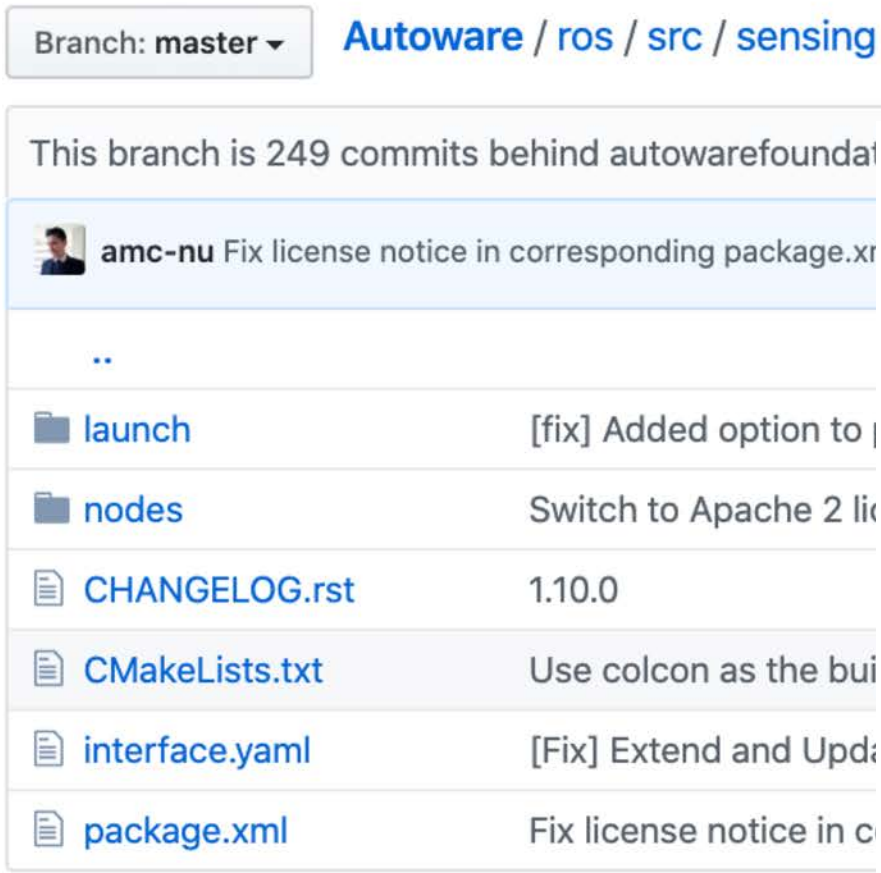
Create new file | Upload files | Find file | History

This branch is 249 commits behind autowarefoundation:master. Pull request Compare

amc-nu Fix license notice in corresponding package.xml — Latest commit `91eb0f8` on Jan 18, 2019

Here are all the packages under the src folder

..		
.config		go
actuation/vehicles/packages		go
common	Fix license notice in corresponding package.xml	12 months ago
computing	Fix license notice in corresponding package.xml	12 months ago
data/packages	Fix license notice in corresponding package.xml	12 months ago
driveworks/packages	Fix license notice in corresponding package.xml	12 months ago
msgs	Fix license notice in corresponding package.xml	12 months ago
sensing	Fix license notice in corresponding package.xml	12 months ago
simulation/lgsvl_simulator_bridge	Fix license notice in corresponding package.xml	12 months ago
socket/packages	Fix license notice in corresponding package.xml	12 months ago
system	Fix license notice in corresponding package.xml	12 months ago
util	Fix license notice in corresponding package.xml	12 months ago



Example of the image processor package in Autoware

Notice: CMakeLists.txt package.xml

The src directory here has been renamed to nodes

Branch: master ▾ **Autoware** / ros / src / sensing / filters / packages / image_processor / **package.xml** Find file | Copy path

amc-nu Fix license notice in corresponding package.xml 91eb0f8 on Jan 18, 2019

6 contributors

22 lines (17 sloc) | 619 Bytes

Image_processor package manifest

```xml
<?xml version="1.0"?>
<package>
  <name>image_processor</name>
  <version>1.10.0</version>
  <description>The image_processor package</description>

  <maintainer email="abrahammonrroy@yahoo.com">amc-nu</maintainer>

  <license>Apache 2</license>

  <buildtool_depend>catkin</buildtool_depend>
  <buildtool_depend>autoware_build_flags</buildtool_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>sensor_msgs</build_depend>
  <build_depend>cv_bridge</build_depend>

  <run_depend>roscpp</run_depend>
  <run_depend>sensor_msgs</run_depend>
  <run_depend>cv_bridge</run_depend>

  <export></export>
</package>
```

Branch: master ▾ **Autoware** / ros / src / sensing / filters / packages / image_processor / nodes / image_rectifier / **image_rectifier_node.cpp**

Find file | Copy path

esteve Switch to Apache 2 license (develop branch) (autowarefoundation#1741) 765397a on Jan 7, 2019

2 contributors

147 lines (117 sloc) | 4.46 KB

Raw | Blame | History

Image_processor cpp source

```
/*
 * Copyright 2017-2019 Autoware Foundation. All rights re
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
//
// Created by amc on 2017-11-15.
//
 */
#include <string>
#include <vector>
#include <ros/ros.h>
#include <cv_bridge/cv_bridge.h>
#include <sensor_msgs/Image.h>
#include <sensor_msgs/image_encodings.h>
#include <sensor_msgs/CameraInfo.h>

#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/calib3d/calib3d.hpp"

#define _NODE_NAME_ "image_rectifier"
```

ROS code is grouped at two different levels:

- Packages
  A named collection of software that is built and treated as an atomic dependency in the ROS build system.
- Stacks
  A named collection of packages for distribution.

source code

header declarations

scripts

message definitions

service definitions

configuration files

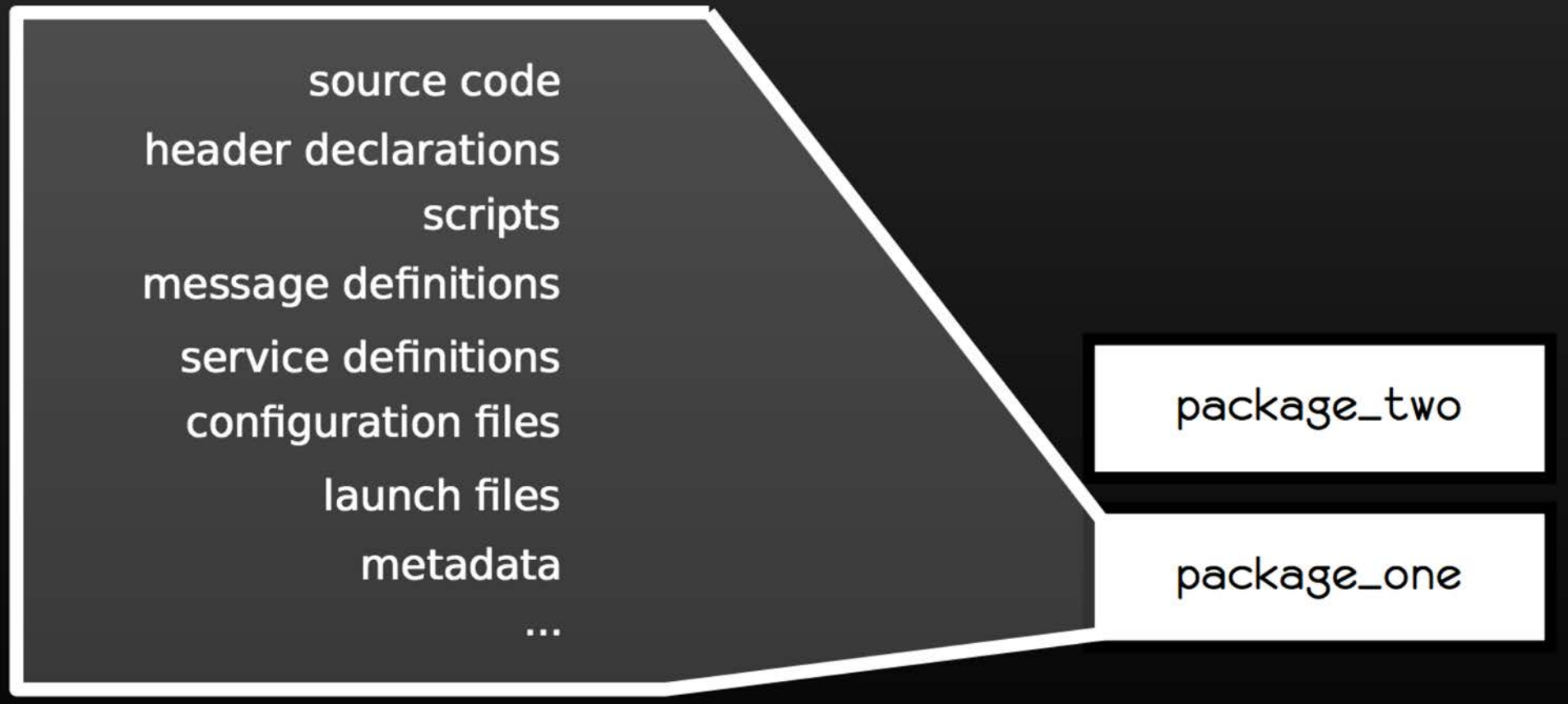
launch files

metadata

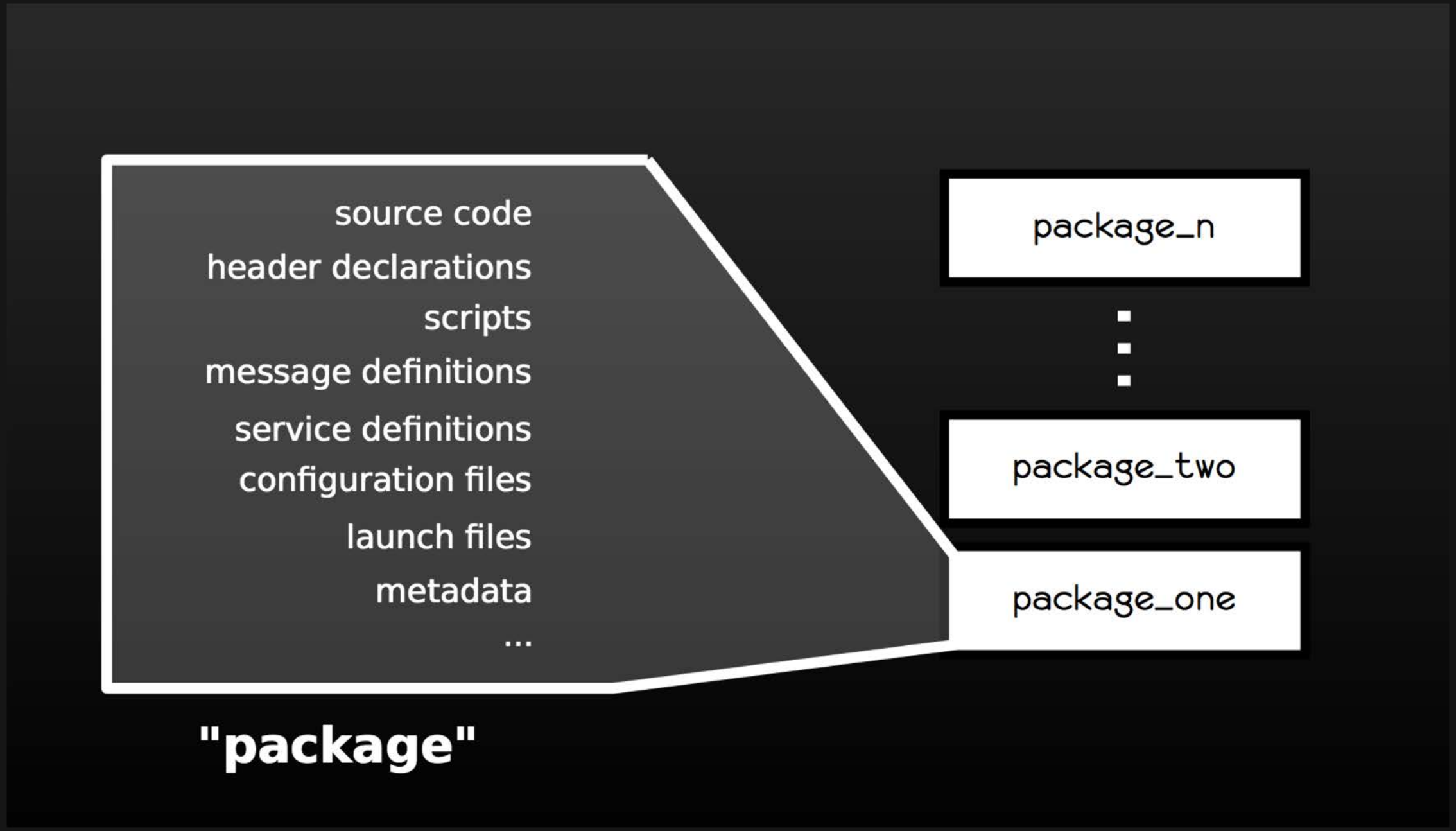
...

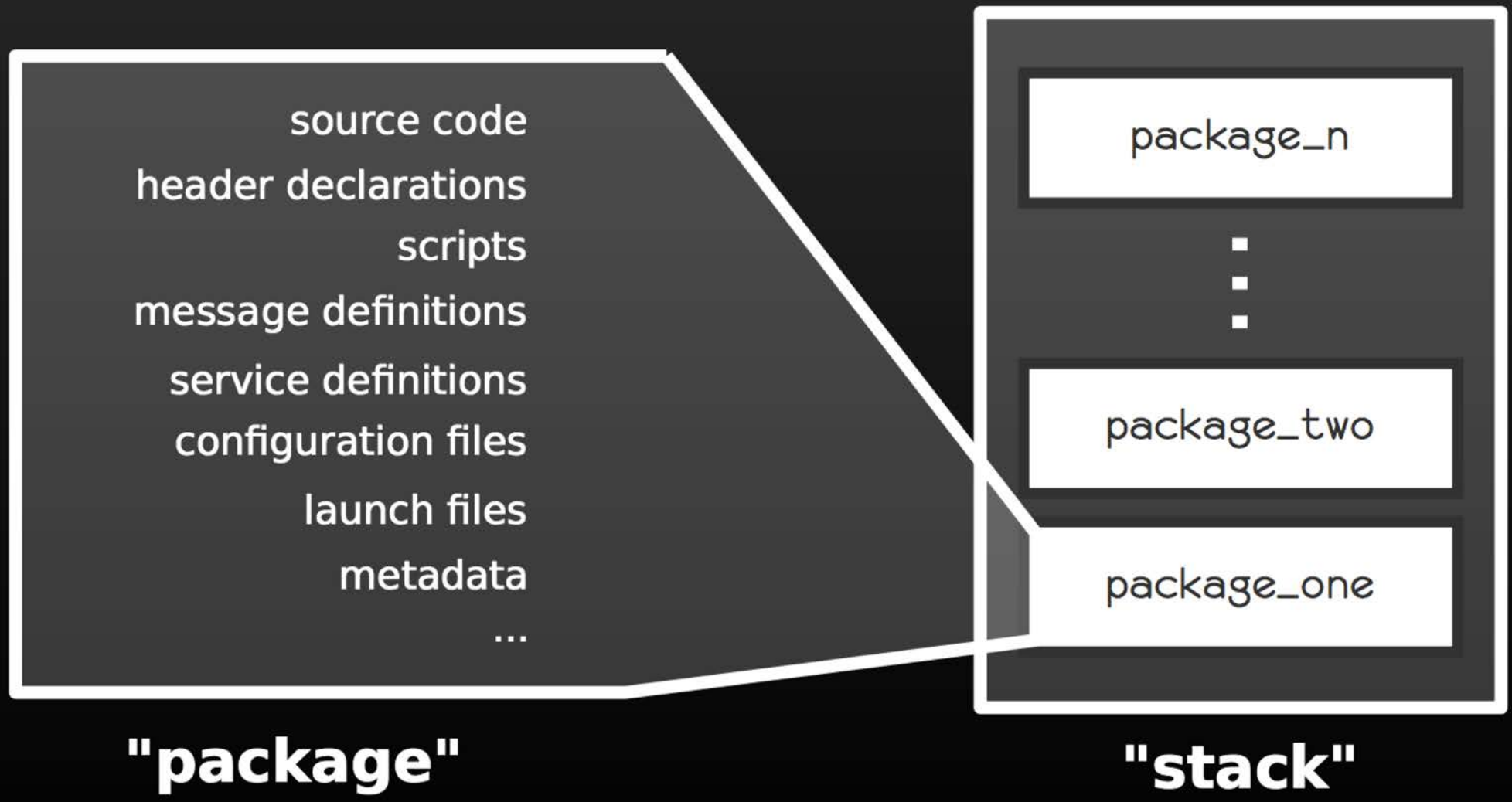

source code
header declarations
scripts
message definitions
service definitions
configuration files
launch files
metadata
...
package_one

"package"


source code
header declarations
scripts
message definitions
service definitions
configuration files
launch files
metadata
...
package_two
package_one

"package"


source code
header declarations
scripts
message definitions
service definitions
configuration files
launch files
metadata
...
"package"
package_n
package_two
package_one

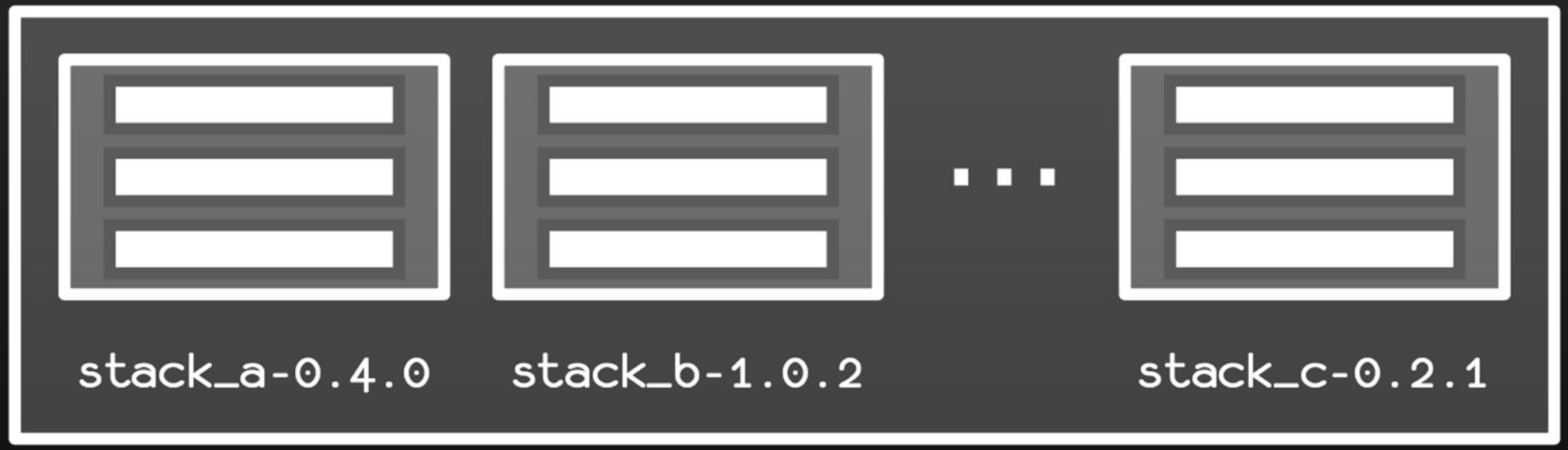

source code
header declarations
scripts
message definitions
service definitions
configuration files
launch files
metadata
...
"package"
package_n
package_two
package_one
"stack"

stack_a-0.4.0

stack_a-0.4.0

stack_b-1.0.2

stack_a-0.4.0 stack_b-1.0.2 stack_c-0.2.1



"distribution"