

# 자율주행 데브코스

## ROS 프로그래밍 기초

---

(주)자이트론

---

허성민

---

smher@xytron.co.kr

---



# Contents

ROS Package 기초

ROS 패키지 만들기

ROS 프로그램의 실행과 검증 확인

Launch 파일 기초

Launch 파일의 tag 활용





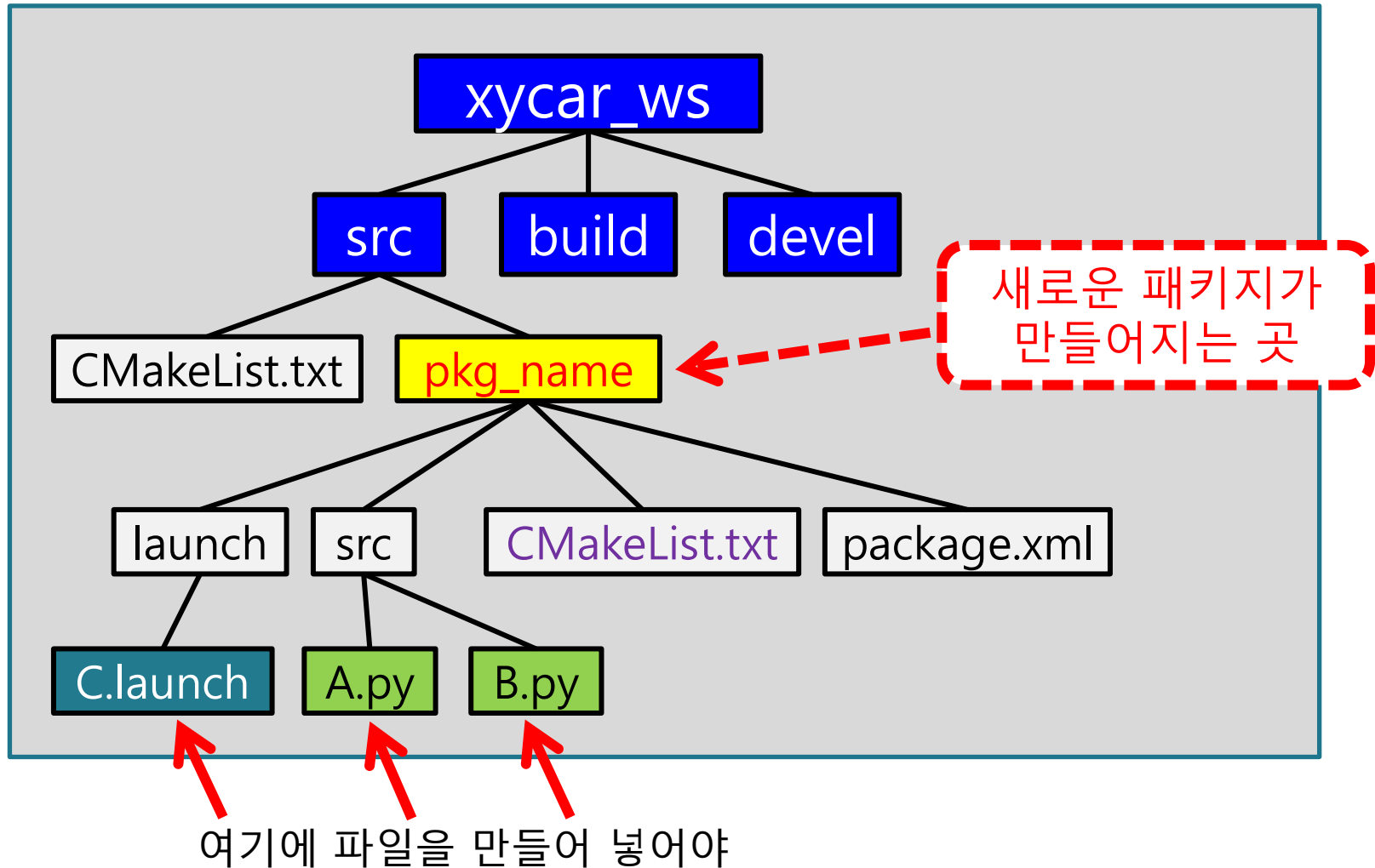
# ROS 프로그래밍 기초

## ROS Package 기초



# ROS 패키지 (Package)

- 패키지 – ROS에서 개발되는 소프트웨어를 논리적 묶음으로 만든 것



# ROS가 제공하는 편리한 명령들

---

- `$ rospack list`
  - 어떤 패키지들이 있는지 나열
- `$ rospack find [package_name]`
  - 이름을 이용해서 패키지 검색
- `$ roscd [location_name[/subdir]]`
  - ROS 패키지 디렉토리로 이동
- `$ rosls [location_name[/subdir]]`
  - Linux ls 와 유사 (경로를 몰라도 이름 적용 가능)
- `$ rosed [file_name]`
  - (환경 설정에 따른) 에디터로 파일을 편집





# ROS 프로그래밍 기초

## ROS 패키지 만들기



# ROS 패키지 만들기

- 패키지를 담을 디렉토리로 이동

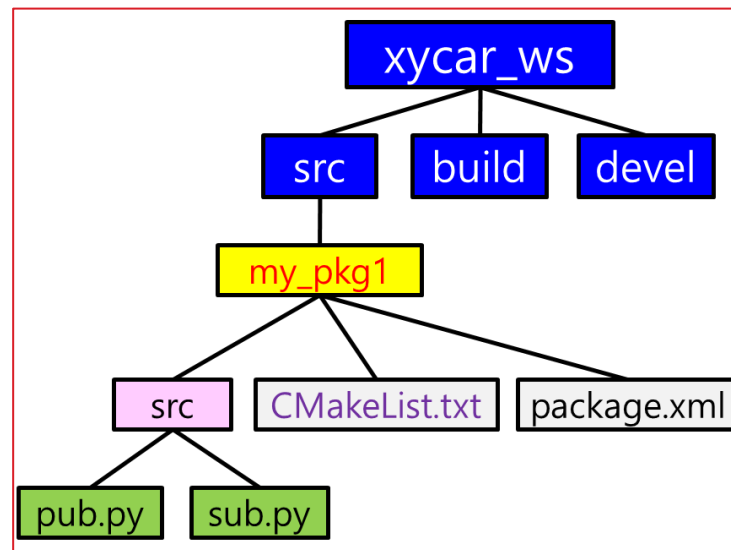
- \$ cd ~/xycar\_ws/src

- 패키지 새로 만들기

- \$ catkin\_create\_pkg my\_pkg1 std\_msgs rospy

패키지 이름

이 패키지가 의존하고 있는  
다른 패키지들을 나열



# ROS 패키지 빌드

- 새로 만든 패키지를 빌드

- \$ cd ~/xycar\_ws
- \$ catkin\_make

'\$ cm' 으로 한번에 실행할 수 있다.

~/.bashrc 파일에서 아래와 같이 alias 선언했기 때문에  
alias cm= ' cd ~/xycar\_ws && catkin\_make '

```
sungmin@machine: ~/xycar_ws
-- Using empy: /usr/bin/empy
-- Using CATKIN_ENABLE_TESTING: ON
-- Call enable_testing()
-- Using CATKIN_TEST_RESULTS_DIR: /home/sungmin/xycar_ws/build/test_results
-- Found gtest sources under '/usr/src/gmock': gtests will be built
-- Found gmock sources under '/usr/src/gmock': gmock will be built
-- Found PythonInterp: /usr/bin/python2 (found version "2.7.12")
-- Using Python nosetests: /usr/bin/nosetests-2.7
-- catkin 0.7.20
-- BUILD_SHARED_LIBS is on
-- BUILD_SHARED_LIBS is on
--
-- traversing 1 packages in topological order:
--   - my_pkg1
--
-- +++ processing catkin package: 'my_pkg1'
-- ==> add_subdirectory(my_pkg1)
-- Configuring done
-- Generating done
-- Build files have been written to: /home/sungmin/xycar_ws/build
####
#### Running command: "make -j2 -l2" in "/home/sungmin/xycar_ws/build"
####
sungmin@machine:~/xycar_ws$
```





# 만들어진 패키지 확인

- `$ rospack find my_pkg1`
- `$ rospack depends1 my_pkg1`
- `$ roscd my_pkg1`

```
sungmin@machine: ~/xycar_ws/src/my_pkg1
sungmin@machine:~/xycar_ws$ rospack find my_pkg1
/home/sungmin/xycar_ws/src/my_pkg1
sungmin@machine:~/xycar_ws$ rospack depends1 my_pkg1
rospy
std_msgs
sungmin@machine:~/xycar_ws$ roscd my_pkg1
sungmin@machine:~/xycar_ws/src/my_pkg1$
```



# 실습 (패키지 만들기)



# 코드 작성 - 프로그래밍

- ~/xycar\_ws/src/**my\_pkg1**/src 위치에, pub.py 라는 이름으로 작성

```
#!/usr/bin/env python
import rospy
from geometry_msgs.msg import Twist

rospy.init_node('my_node', anonymous=True)
pub = rospy.Publisher('/turtle1/cmd_vel', Twist, queue_size=10)

msg = Twist()
msg.linear.x = 2.0
msg.linear.y = 0.0
msg.linear.z = 0.0
msg.angular.x = 0.0
msg.angular.y = 0.0
msg.angular.z = 1.8

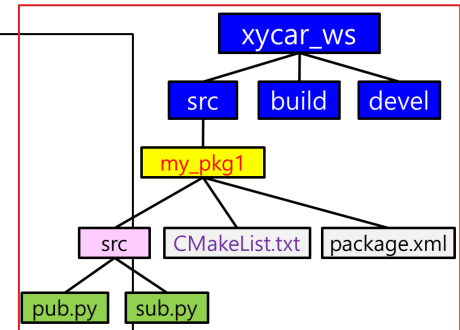
rate = rospy.Rate(1)

while not rospy.is_shutdown():
    pub.publish(msg)
    rate.sleep()
```

노드를 만들고

Publisher 객체 생성

1Hz 주기로 메시지 발행  
(1초에 한번씩 발행)



# 프로그램 실행 권한

- 작성한 파이썬 코드를 실행시키려면 **실행권한**이 있어야 한다.
- 다음과 같은 방법으로 실행권한을 부여해야 한다.
  - `$ chmod +x pub.py`
- `$ ls -l` 명령으로 실행권한 여부를 확인할 수 있다.

주의 !

```
sungmin@machine: ~/xycar_ws/src/my_pkg1/src
sungmin@machine:~/xycar_ws/src/my_pkg1/src$ ls -l
total 4
-rw-rw-r-- 1 sungmin sungmin 510 12월 14 2019 pub.py

sungmin@machine:~/xycar_ws/src/my_pkg1/src$ chmod +x pub.py

sungmin@machine:~/xycar_ws/src/my_pkg1/src$ ls -l
total 4
-rwxrwxr-x 1 sungmin sungmin 510 12월 14 2019 pub.py

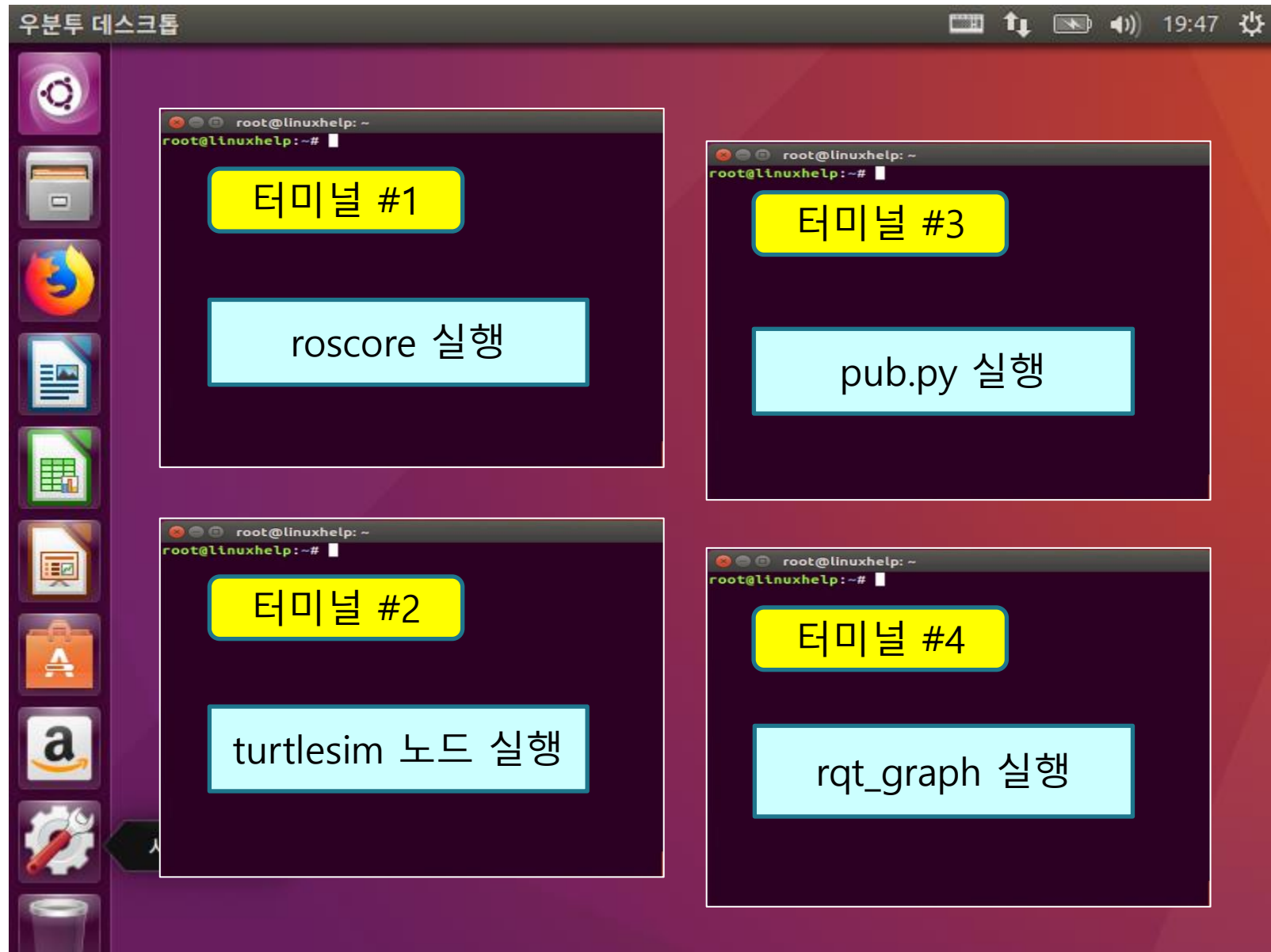
sungmin@machine:~/xycar_ws/src/my_pkg1/src$
```

실행권한이 없다

\$ chmod 명령으로  
실행권한이 생겼다.

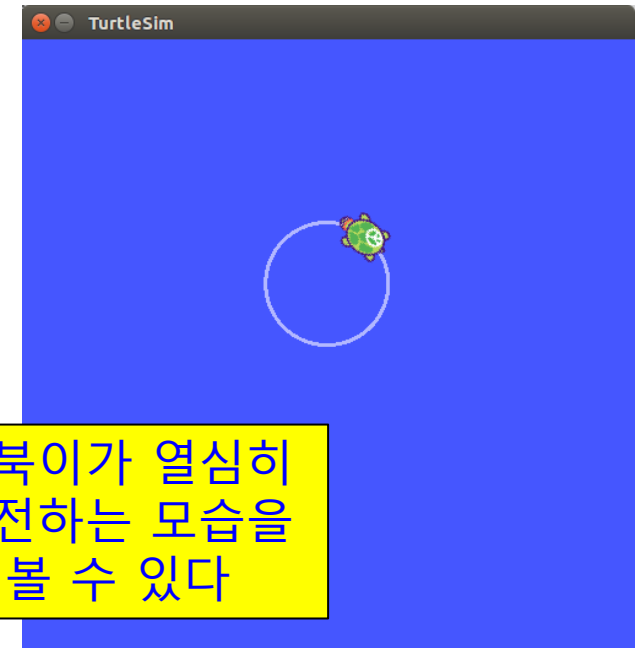


# 프로그램 실행과 확인



# 프로그램 실행과 확인

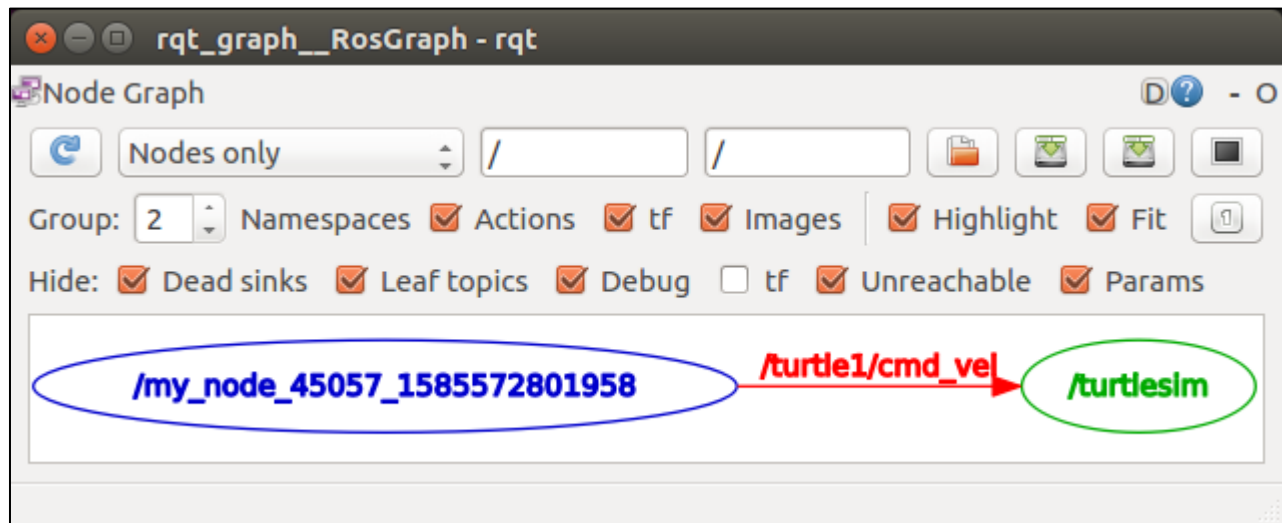
- 각각 다른 터미널에서 roscore와 turtlesim\_node를 실행
  - 터미널 #1에서
    - ▶ `$ roscore`
  - 터미널 #2에서
    - ▶ `$ roslaunch turtlesim turtlesim_node`
- 조금 전 만든 pub.py 프로그램을 실행시킨다
  - 터미널 #3에서
    - ▶ `$ chmod +x pub.py`
    - ▶ `$ roslaunch my_pkg1 pub.py`



거북이가 열심히  
회전하는 모습을  
볼 수 있다

# 내가 만든 노드가 잘 동작하고 있는가?

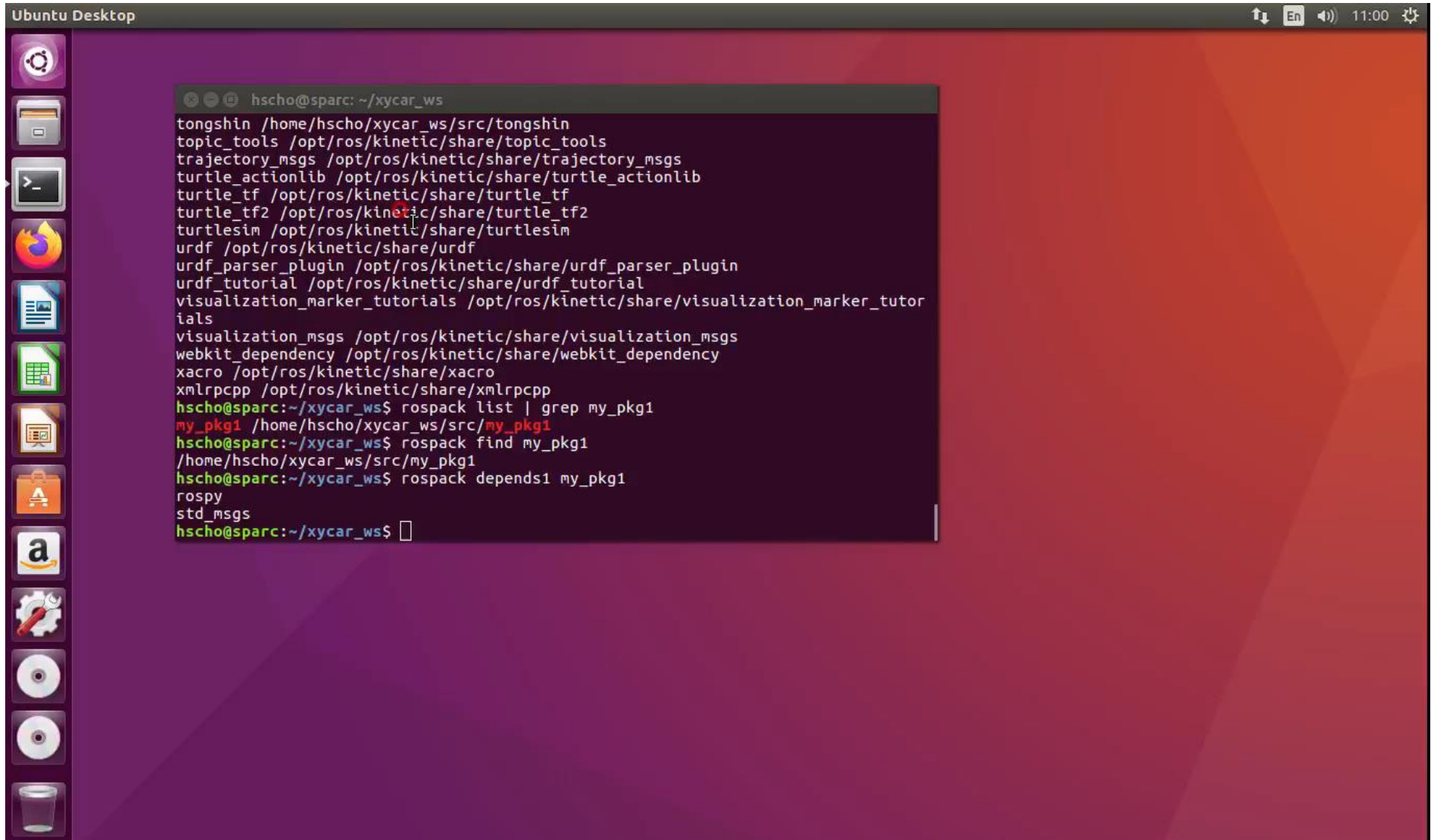
- 노드의 상태를 몇 가지 ROS 명령으로 확인할 수 있다
  - `$ rqt_graph`



- `$ rostopic list`

```
Terminal
xycar:~$ rostopic list
/my_node_4474_1594167230191
/rosout
/turtlesim
xycar:~$
```

# 실습(노드 만들기)



The image shows an Ubuntu Desktop environment with a terminal window open. The terminal displays the following content:

```
hscho@sparc: ~/xycar_ws
tongshin /home/hscho/xycar_ws/src/tongshin
topic_tools /opt/ros/kinetic/share/topic_tools
trajectory_msgs /opt/ros/kinetic/share/trajectory_msgs
turtle_actionlib /opt/ros/kinetic/share/turtle_actionlib
turtle_tf /opt/ros/kinetic/share/turtle_tf
turtle_tf2 /opt/ros/kinetic/share/turtle_tf2
turtlesim /opt/ros/kinetic/share/turtlesim
urdf /opt/ros/kinetic/share/urdf
urdf_parser_plugin /opt/ros/kinetic/share/urdf_parser_plugin
urdf_tutorial /opt/ros/kinetic/share/urdf_tutorial
visualization_marker_tutorials /opt/ros/kinetic/share/visualization_marker_tutorials
visualization_msgs /opt/ros/kinetic/share/visualization_msgs
webkit_dependency /opt/ros/kinetic/share/webkit_dependency
xacro /opt/ros/kinetic/share/xacro
xmlrpcpp /opt/ros/kinetic/share/xmlrpcpp
hscho@sparc:~/xycar_ws$ rospack list | grep my_pkg1
my_pkg1 /home/hscho/xycar_ws/src/my_pkg1
hscho@sparc:~/xycar_ws$ rospack find my_pkg1
/home/hscho/xycar_ws/src/my_pkg1
hscho@sparc:~/xycar_ws$ rospack depends1 my_pkg1
rospy
std_msgs
hscho@sparc:~/xycar_ws$
```



# 이번에는 구독자(Subscriber)를 살펴보자

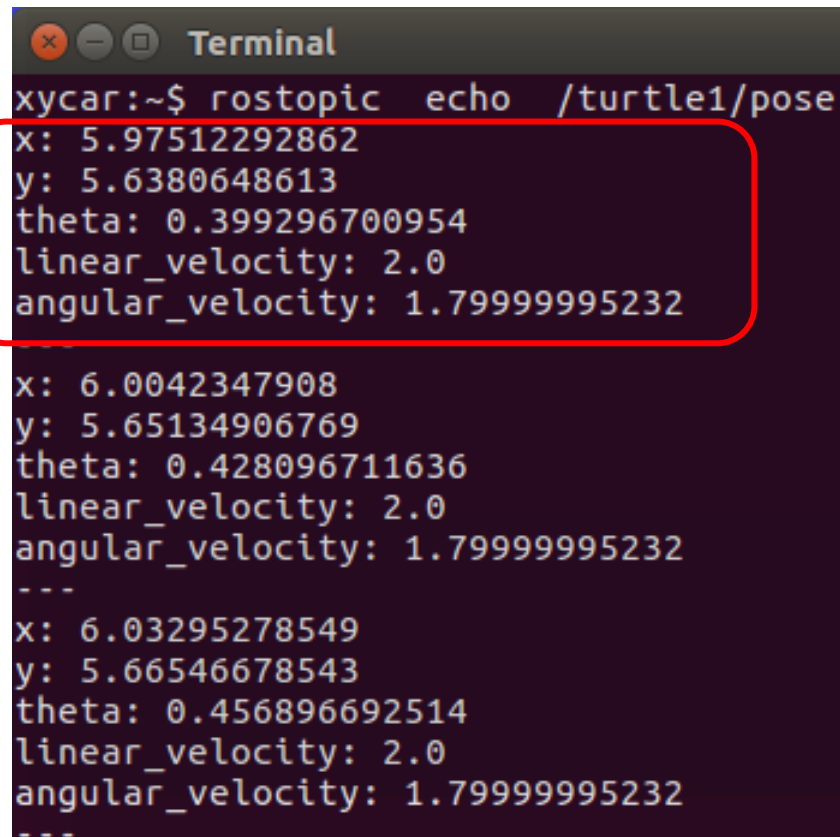
- 우선, turtle이 어떤 토픽에 어떤 메시지를 발행하고 있는지 알아보자
  - `$ rostopic list`
  - `$ rostopic type /turtle1/pose`
  - `$ rosmmsg show turtlesim/Pose`
  - `$ rostopic echo /turtle1/pose`

```
Terminal
xycar:~$ rostopic list
/rosout
/rosout_agg
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
xycar:~$ rostopic type /turtle1/pose
turtlesim/Pose
xycar:~$ rosmmsg show turtlesim/Pose
float32 x
float32 y
float32 theta
float32 linear_velocity
float32 angular_velocity
```



# 이번에는 구독자(Subscriber)를 살펴보자

- 토픽에 어떤 메시지를 발행하고 있는지 알아보자
  - `$ rostopic echo /turtle1/pose`



A terminal window titled "Terminal" with a dark background. The command `xycar:~$ rostopic echo /turtle1/pose` has been entered. The output shows a series of turtle1 pose messages. The first message is highlighted with a red rounded rectangle. The messages are as follows:

```
xycar:~$ rostopic echo /turtle1/pose
x: 5.97512292862
y: 5.6380648613
theta: 0.399296700954
linear_velocity: 2.0
angular_velocity: 1.79999995232
---
x: 6.0042347908
y: 5.65134906769
theta: 0.428096711636
linear_velocity: 2.0
angular_velocity: 1.79999995232
---
x: 6.03295278549
y: 5.66546678543
theta: 0.456896692514
linear_velocity: 2.0
angular_velocity: 1.79999995232
---
```



# 코드 작성 - 프로그래밍

- ~/xycar\_ws/src/my\_pkg1/src 위치에,

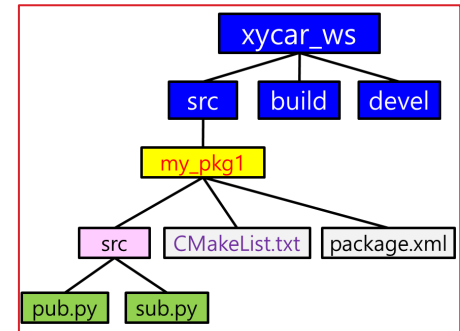
아래와 같은 프로그램을 sub.py 라는 이름으로 작성

```
#!/usr/bin/env python

import rospy
from turtlesim.msg import Pose

def callback(data):
    s = "Location: %.2f, %.2f" % (data.x, data.y)
    rospy.loginfo(rospy.get_caller_id() + s)

rospy.init_node("my_listener", anonymous=True)
rospy.Subscriber("/turtle1/pose", Pose, callback)
rospy.spin()
```

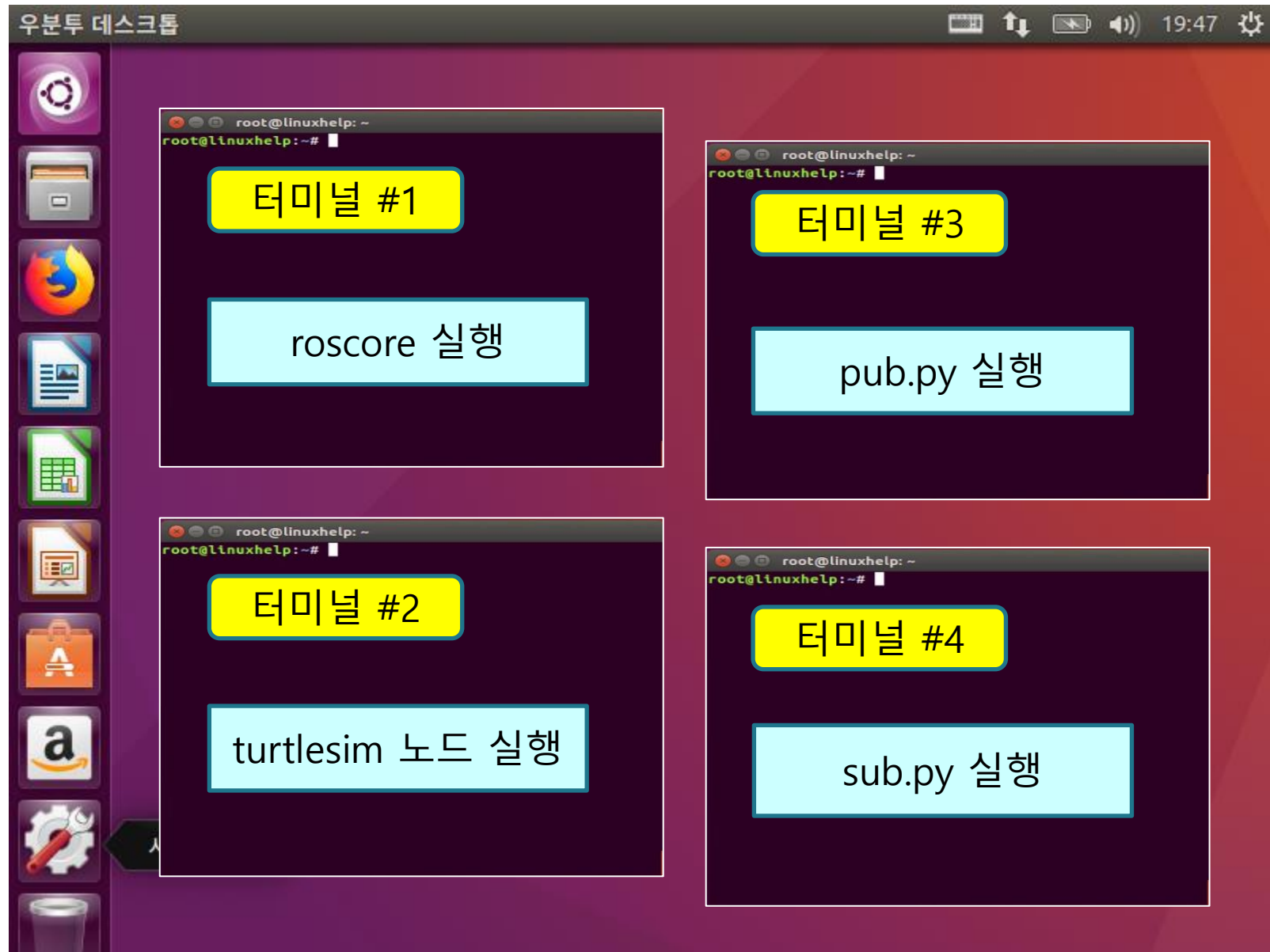


노드를 만들고

Subscriber 객체 생성

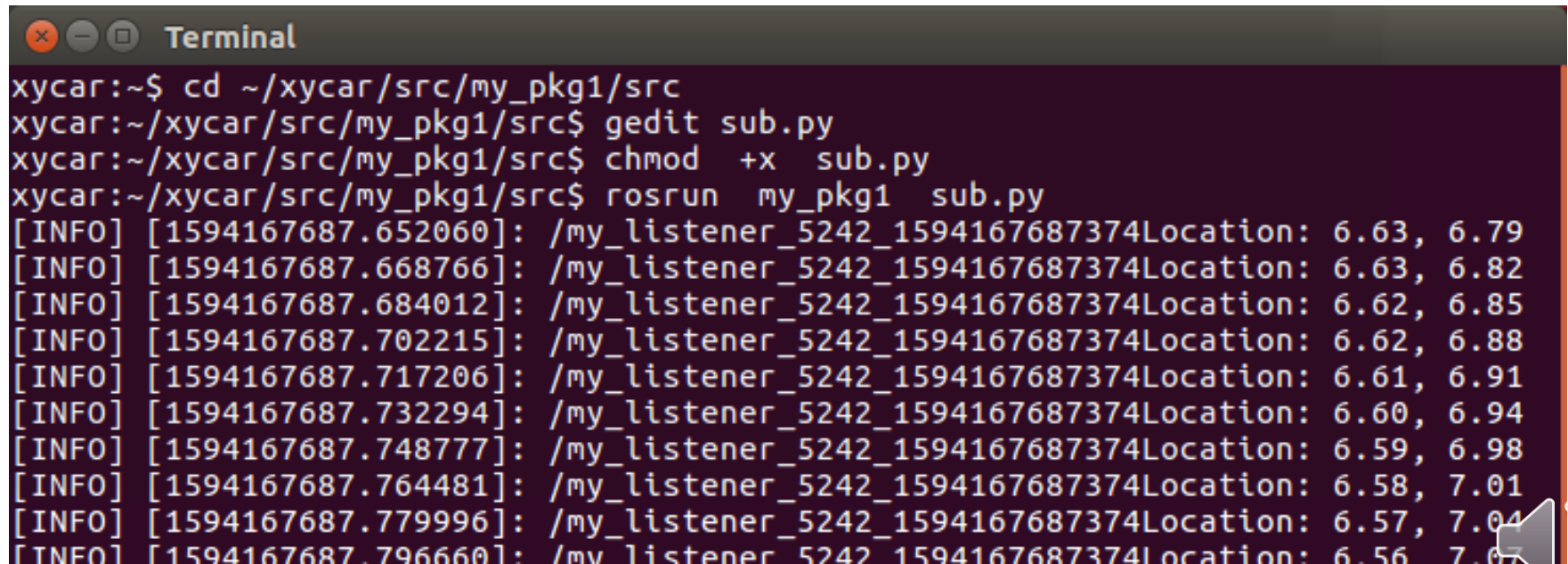
메시지를 수신하면 이 함수가 호출됨

# 프로그램 실행과 확인



# 프로그램 실행과 확인

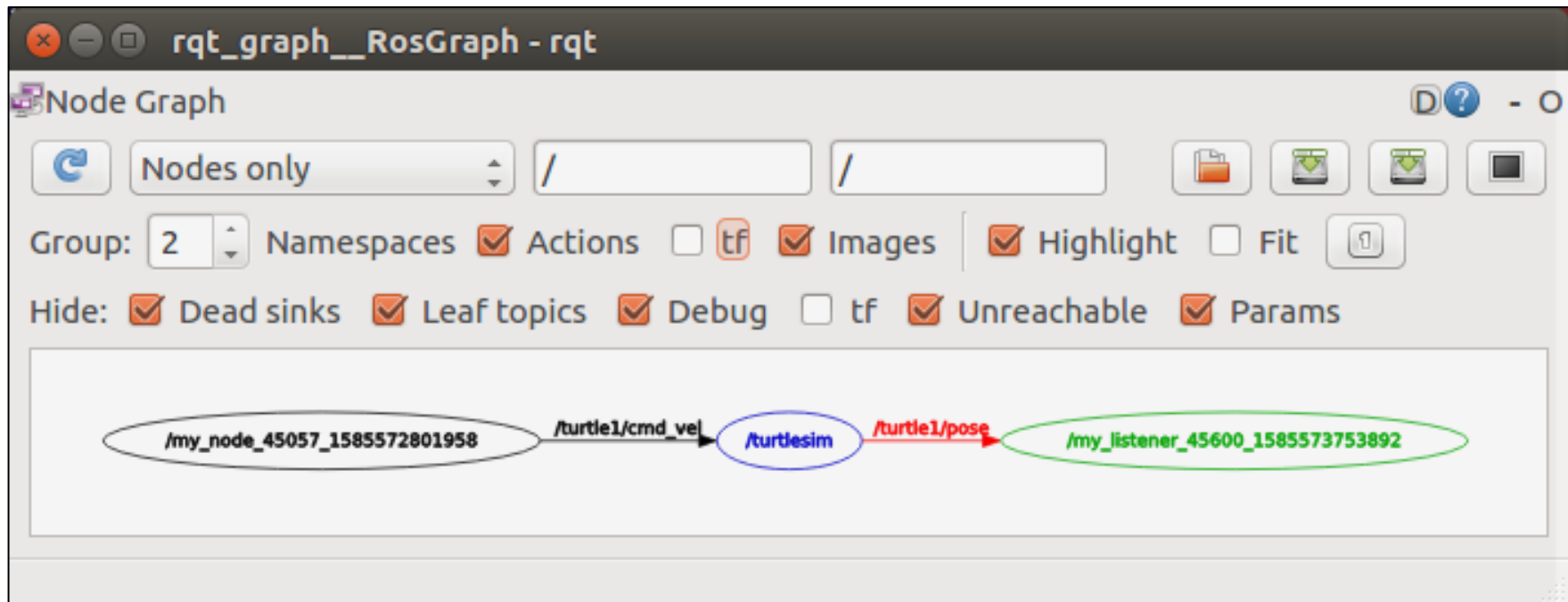
- 우선 앞서의 turtlesim\_node와 pub.py가 실행되고 있는 상태에서
- 조금 전 만든 sub.py 프로그램을 실행시킨다
  - `$ chmod +x sub.py`
  - `$ rosrun my_pkg1 sub.py`

A terminal window titled "Terminal" with a dark background and light-colored text. It shows the execution of a ROS program. The user navigates to the directory ~/xycar/src/my\_pkg1/src, creates a file sub.py with gedit, sets it to be executable with chmod, and then runs it with rosrun. The output shows a series of INFO messages from a listener, each reporting a location (x, y) that changes over time.

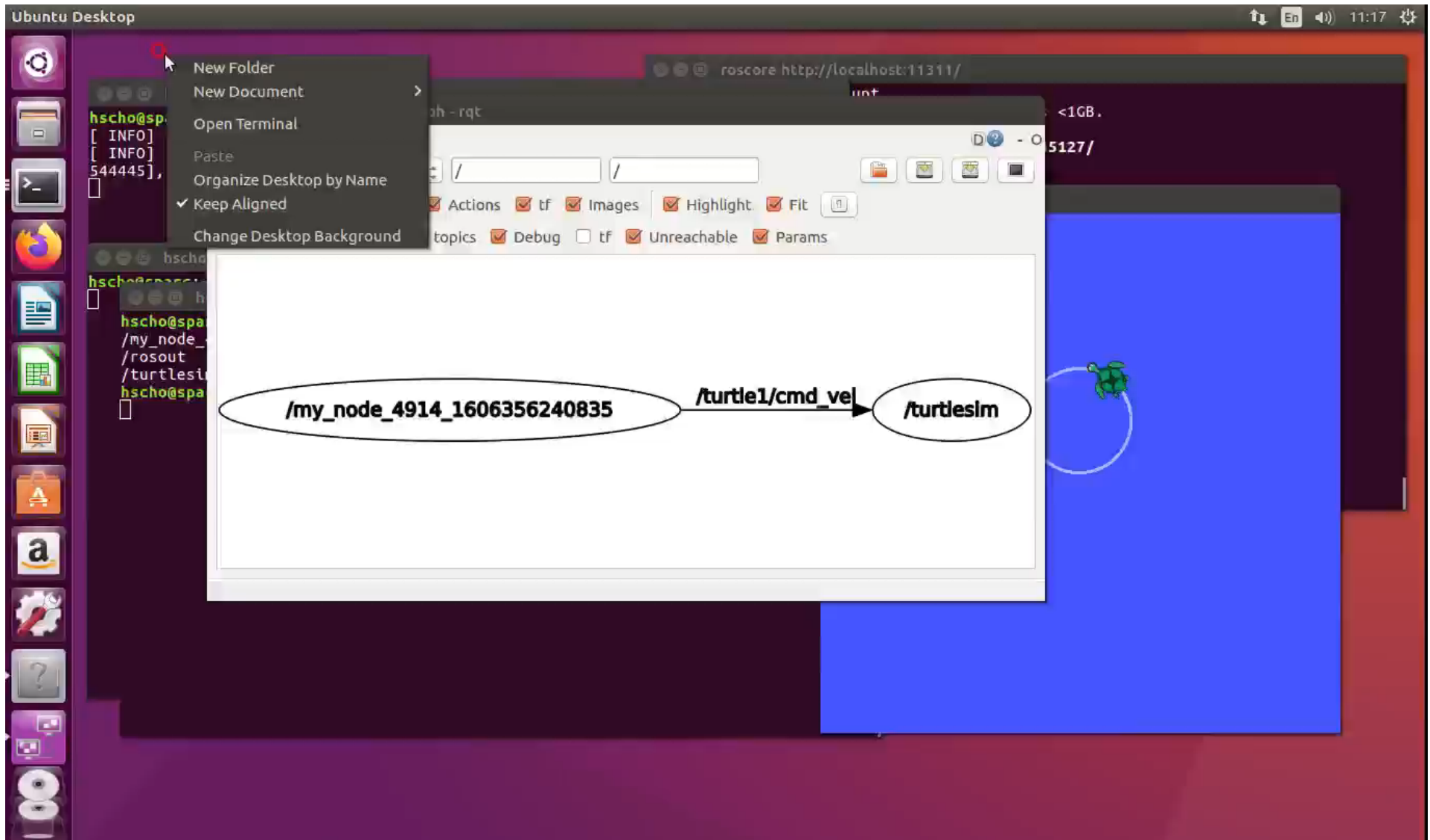
```
xy-car:~$ cd ~/xycar/src/my_pkg1/src
xy-car:~/xycar/src/my_pkg1/src$ gedit sub.py
xy-car:~/xycar/src/my_pkg1/src$ chmod +x sub.py
xy-car:~/xycar/src/my_pkg1/src$ rosrun my_pkg1 sub.py
[INFO] [1594167687.652060]: /my_listener_5242_1594167687374Location: 6.63, 6.79
[INFO] [1594167687.668766]: /my_listener_5242_1594167687374Location: 6.63, 6.82
[INFO] [1594167687.684012]: /my_listener_5242_1594167687374Location: 6.62, 6.85
[INFO] [1594167687.702215]: /my_listener_5242_1594167687374Location: 6.62, 6.88
[INFO] [1594167687.717206]: /my_listener_5242_1594167687374Location: 6.61, 6.91
[INFO] [1594167687.732294]: /my_listener_5242_1594167687374Location: 6.60, 6.94
[INFO] [1594167687.748777]: /my_listener_5242_1594167687374Location: 6.59, 6.98
[INFO] [1594167687.764481]: /my_listener_5242_1594167687374Location: 6.58, 7.01
[INFO] [1594167687.779996]: /my_listener_5242_1594167687374Location: 6.57, 7.04
[INFO] [1594167687.796660]: /my_listener_5242_1594167687374Location: 6.56, 7.07
```

# 내가 만든 노드가 잘 동작하고 있는가?

- 노드가 잘 동작하고 있는지 확인
  - `$ rqt_graph`



# 실습 (구독자 노드)



The screenshot shows an Ubuntu Desktop environment with the following elements:

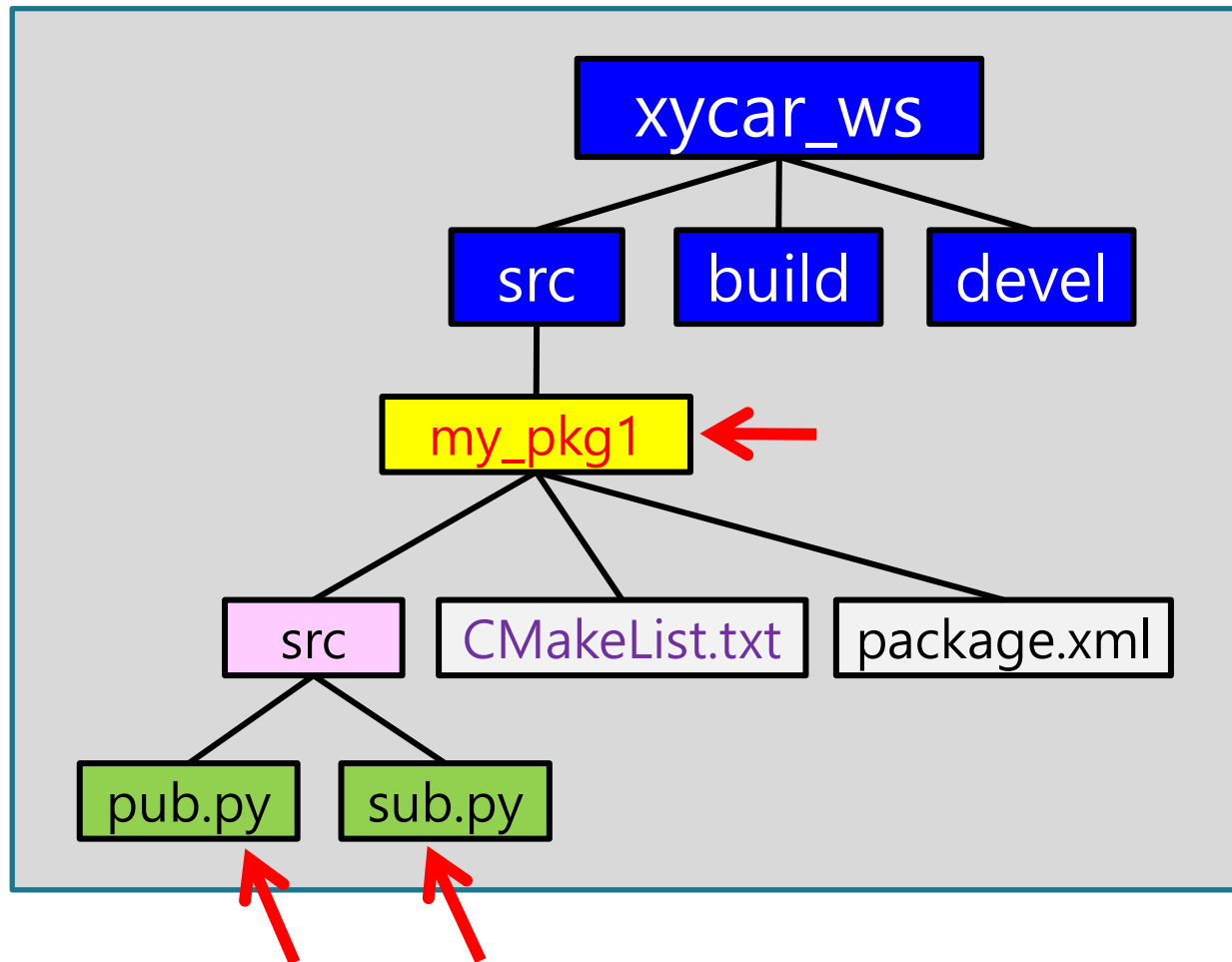
- Terminal Window:** Displays the command `roscore http://localhost:11311/` and the output `[ INFO] 544445],`.
- File Explorer Window:** Shows a directory structure with the following contents:
  - `/my_node_4914_1606356240835`
  - `/rosout`
  - `/turtlesim`
- Diagram:** A central diagram illustrating the node connection:
 

```

      graph LR
      A([/my_node_4914_1606356240835]) --> B[/turtle1/cmd_vel]
      B --> C([/turtlesim])
      
```
- Background:** A blue square representing the `/turtlesim` environment, with a small green turtle icon visible on the right side.

# 정리하면

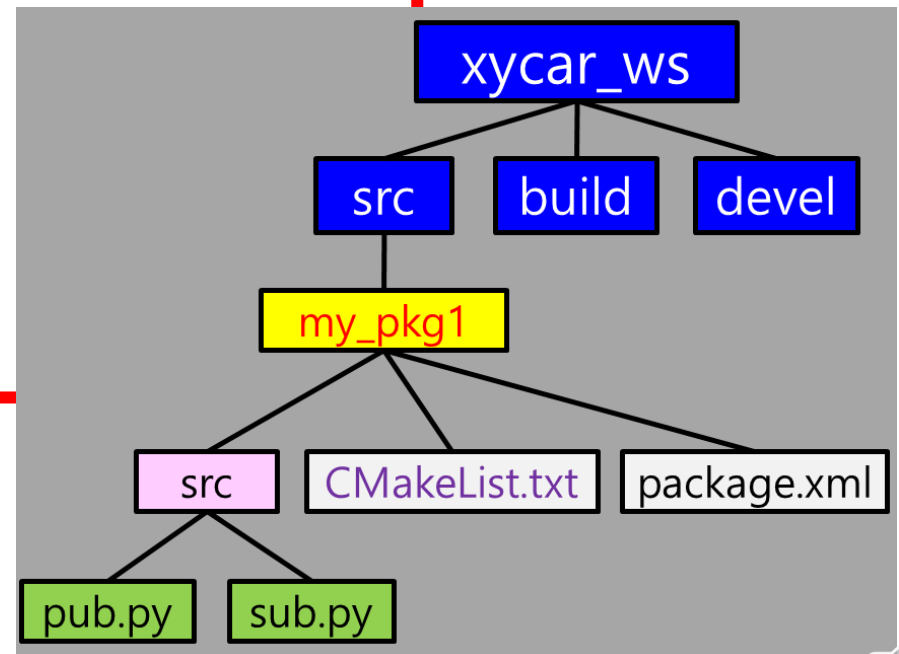
- my\_pkg1 패키지를 만들고 pub.py, sub.py, 2개 파일을 작성





# 정리하면

```
$ cd ~/xycar_ws/src
$ catkin_create_pkg my_pkg1 std_msgs rospy
$ cm
$ cd ~/xycar_ws/src/my_pkg1/src
$ gedit sub.py
$ gedit pub.py
$ chmod +x sub.py pub.py
$ rosrun my_pkg1 pub.py
$ rosrun my_pkg1 sub.py
```





# ROS 프로그래밍 기초

## Launch 파일 사용하기



# ROS의 명령어 – roslaunch

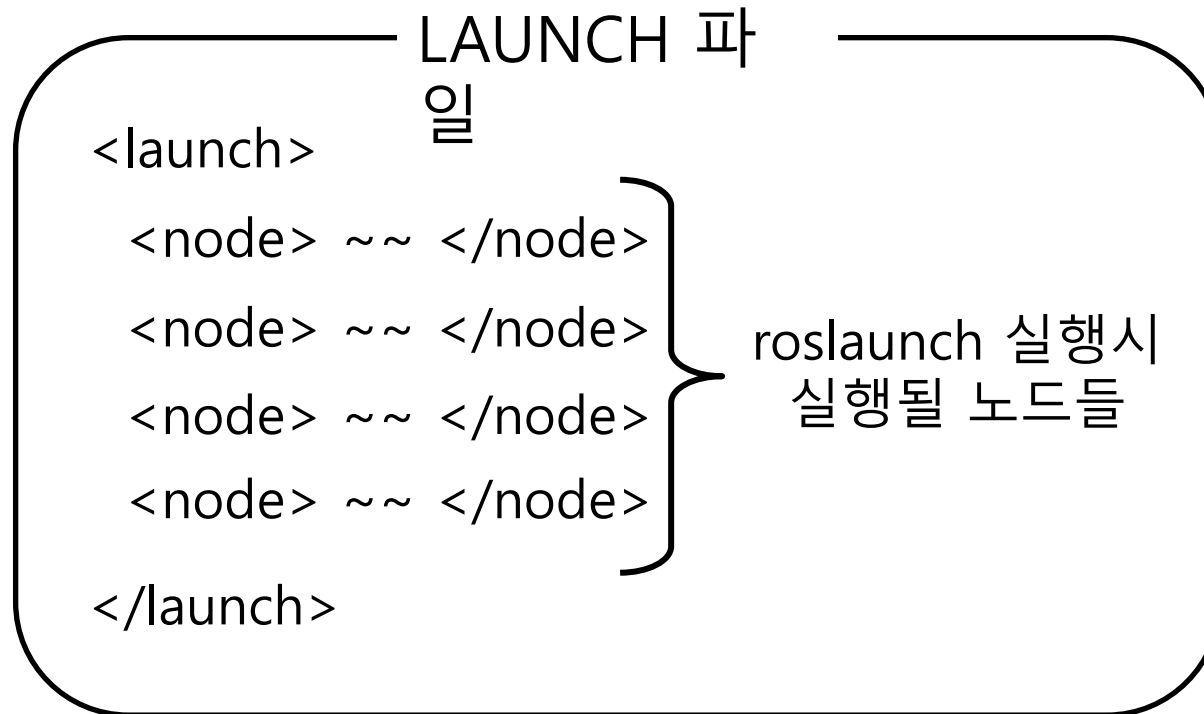
- \$ roslaunch
  - \*.launch 파일 내용에 따라 여러 노드들을 한꺼번에 실행시킬 수 있음

```
nvidia@tegra-ubuntu: ~  
nvidia@tegra-ubuntu:~$ roslaunch  
Usage: roslaunch [options] [package] <filename> [arg_name:=value...]  
       roslaunch [options] <filename> [<filename>...] [arg_name:=value...]  
  
If <filename> is a single dash ('-'), launch XML is read from standard input.  
roslaunch: error: you must specify at least one input file
```

- roslaunch 사용법
  - \$ roslaunch [패키지이름] [실행시킬 launch 파일 이름]
  - 사용 사례
    - ▶ \$ roslaunch my\_pkg1 aaa.launch
  - 이때 [실행시킬 launch 파일]은 반드시 패키지에 포함된 launch 파일이어야 함.

## \*.launch 파일

- roslaunch 명령어를 이용하여 많은 노드를 동시에 실행시키기 위한 파일
- 실행시킬 노드들의 정보가 XML 형식으로 기록되어 있음.



# \*.launch에서 사용하는 Tag : node

- node 태그

- 실행할 노드 정보를 입력할 때 사용되는 태그

```
<node pkg="패키지 명" type="노드가 포함된 소스파일 명" name="노드 이름" />
```

- 속성

예시 : `<node pkg="my_pkg1" type="pub.py" name="pub_node"/>`

- ▶ pkg : 실행시킬 노드의 패키지 이름을 입력하는 속성.

- 반드시 빌드된 패키지의 이름을 입력해야 함.

- ▶ type : 노드의 소스코드가 담긴 파이썬 파일의 이름을 입력하는 속성.

- 이때 파이썬 .py 파일은 반드시 실행권한이 있어야 함.

실행권한 없을 시 다음 에러 발생

```
ERROR: cannot launch node of type [ 패키지명 / 소스파일명 ] :  
can't locate node [ 소스파일명 ] in package [ 패키지명 ]
```

- ▶ name : 노드의 이름을 입력하는 속성.

- 소스코드에서 지정한 노드의 이름을 무시하고, launch 파일에 기록된 노드의 이름으로 노드가 실행됨.



# \*.launch에서 사용하는 Tag : include

- include 태그
  - 다른 launch 파일을 불러오고 싶을 때 사용하는 태그

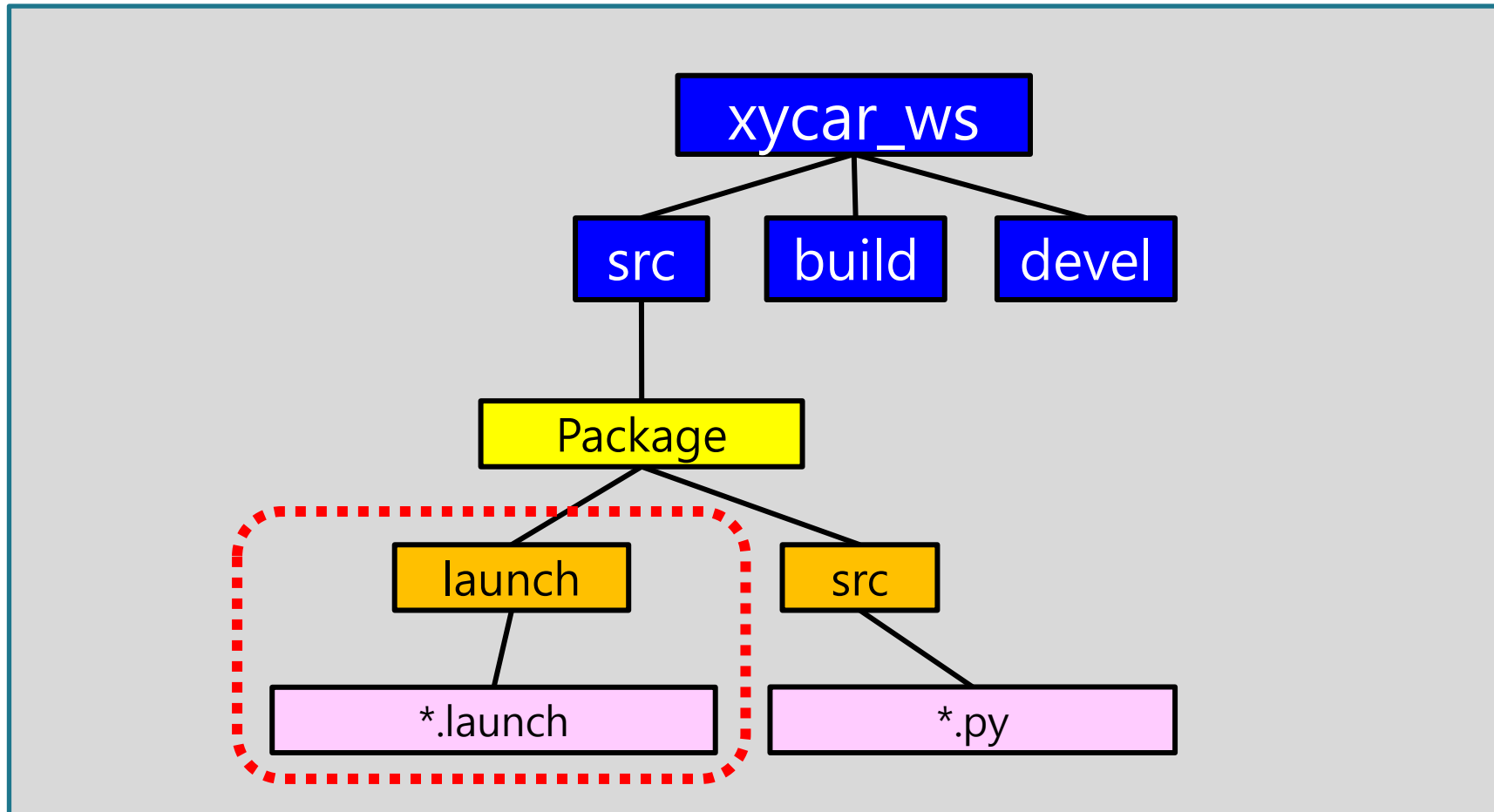
```
<include file= " 같이 실행할 *.launch 파일 경로 " />
```

```
예시 : <include file="../../../cam/cam_test.launch" />
```

```
예시 : <include file="$(find usb_cam)/src/launch/aaa.launch"  
/>
```

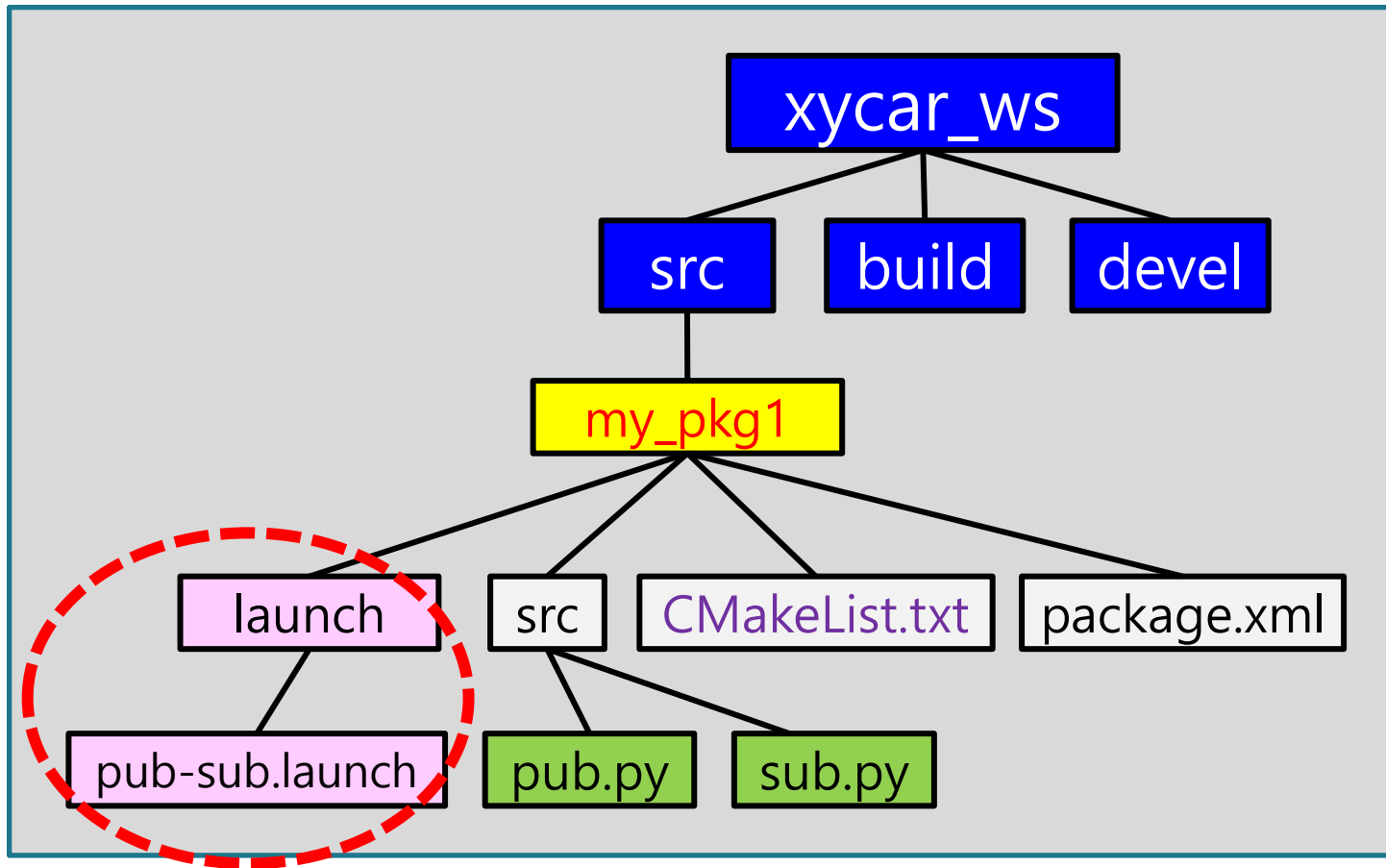
- 속성
  - ▶ file : 함께 실행시킬 \*.launch파일의 경로를 입력하는 속성.

## \*.launch 파일의 위치



# Launch 파일 생성

- 패키지 디렉토리 아래에 'launch' 라는 디렉토리 만들고
  - 그 안에 .launch 확장자의 파일을 만들어야 함





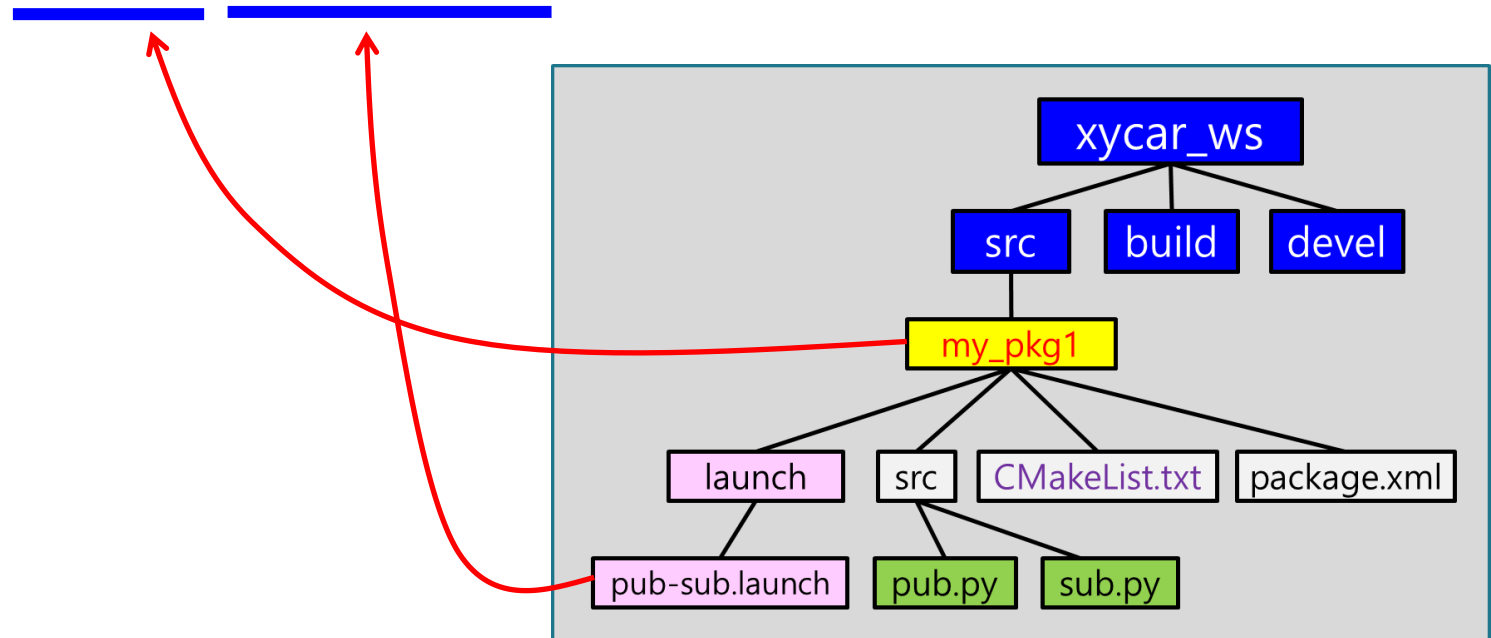
# Launch 파일 작성

- 여러 노드를 함께 실행시킬 때 편리
  - `$ roslaunch [package_name] [file.launch]`
  - 다수의 노드를 한꺼번에 실행 가능
  - 파라미터 값을 노드에 전달 가능
- Launch 파일 작성
  - `$ gedit pub-sub.launch`

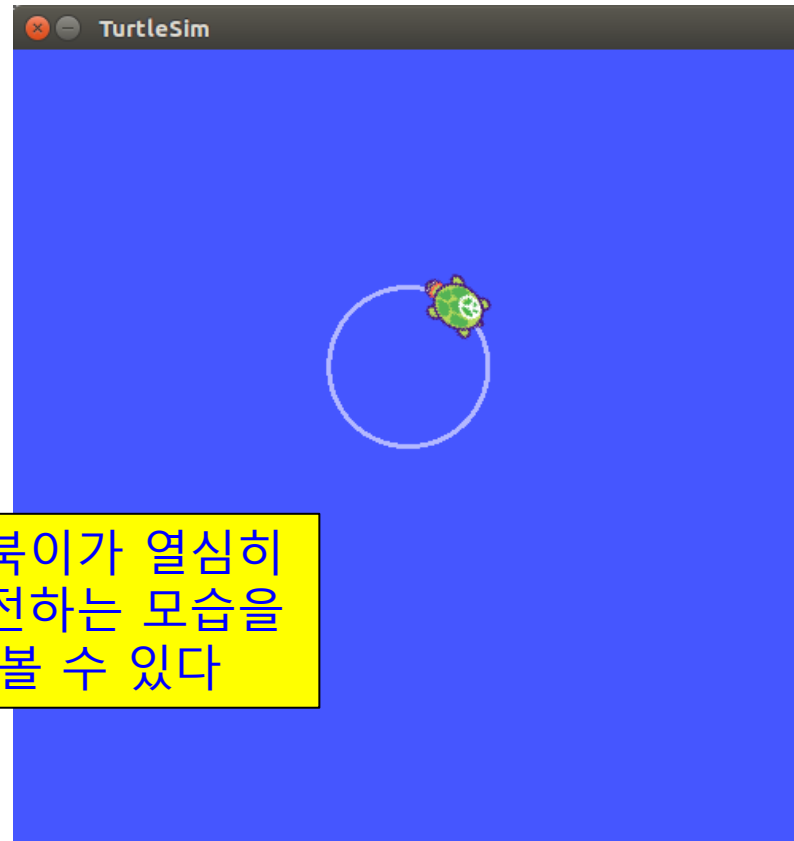
```
<launch>
  <node pkg="turtlesim" type="turtlesim_node" name="turtlesim_node"/>
  <node pkg="my_pkg1" type="pub.py" name="pub_node"/>
  <node pkg="my_pkg1" type="sub.py" name="sub_node" output="screen"/>
</launch>
```

# Launch 파일 실행

- Launch 파일 실행 방법
  - `$ roslaunch my_pkg1 pub-sub.launch`



- `$ roslaunch` 명령을 사용할 때는
  - 별도로 `$ roscore` 명령을 실행할 필요가 없다.
  - 내부적으로 `roscore`가 자동으로 실행된다. → 매우 편하다.

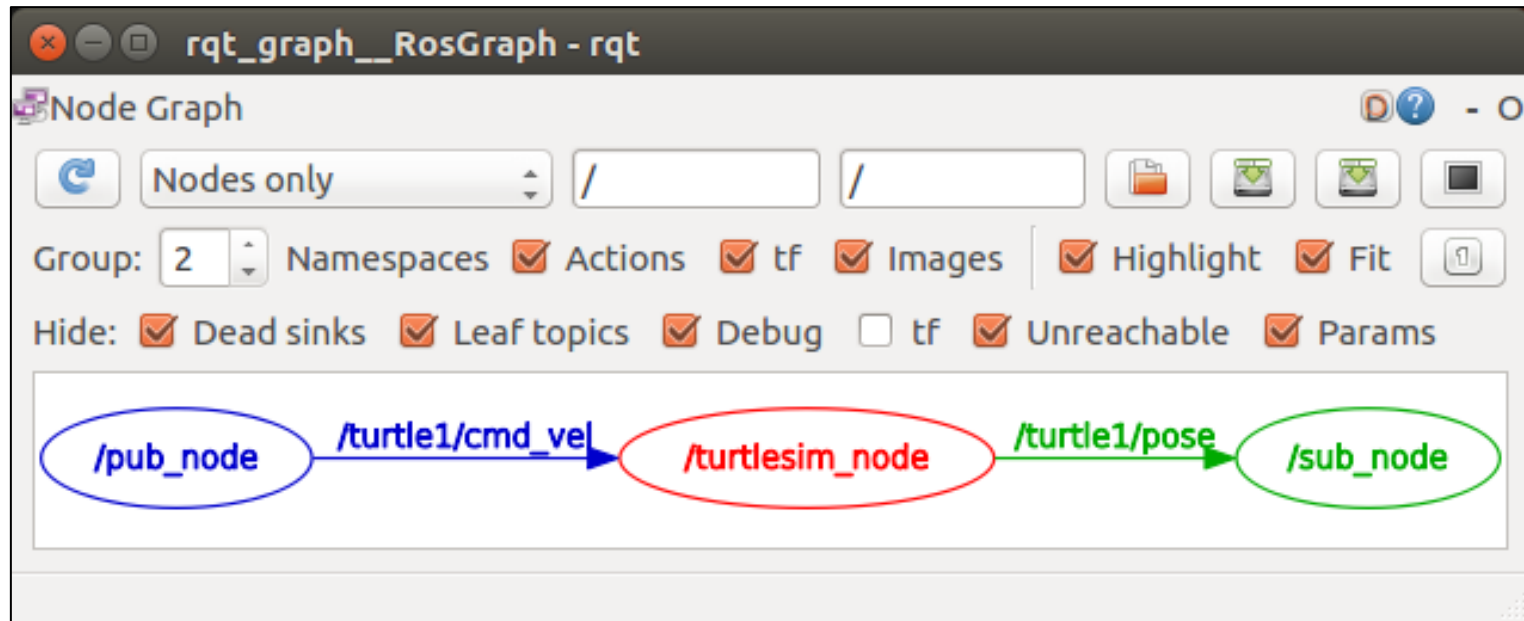


거북이가 열심히  
회전하는 모습을  
볼 수 있다

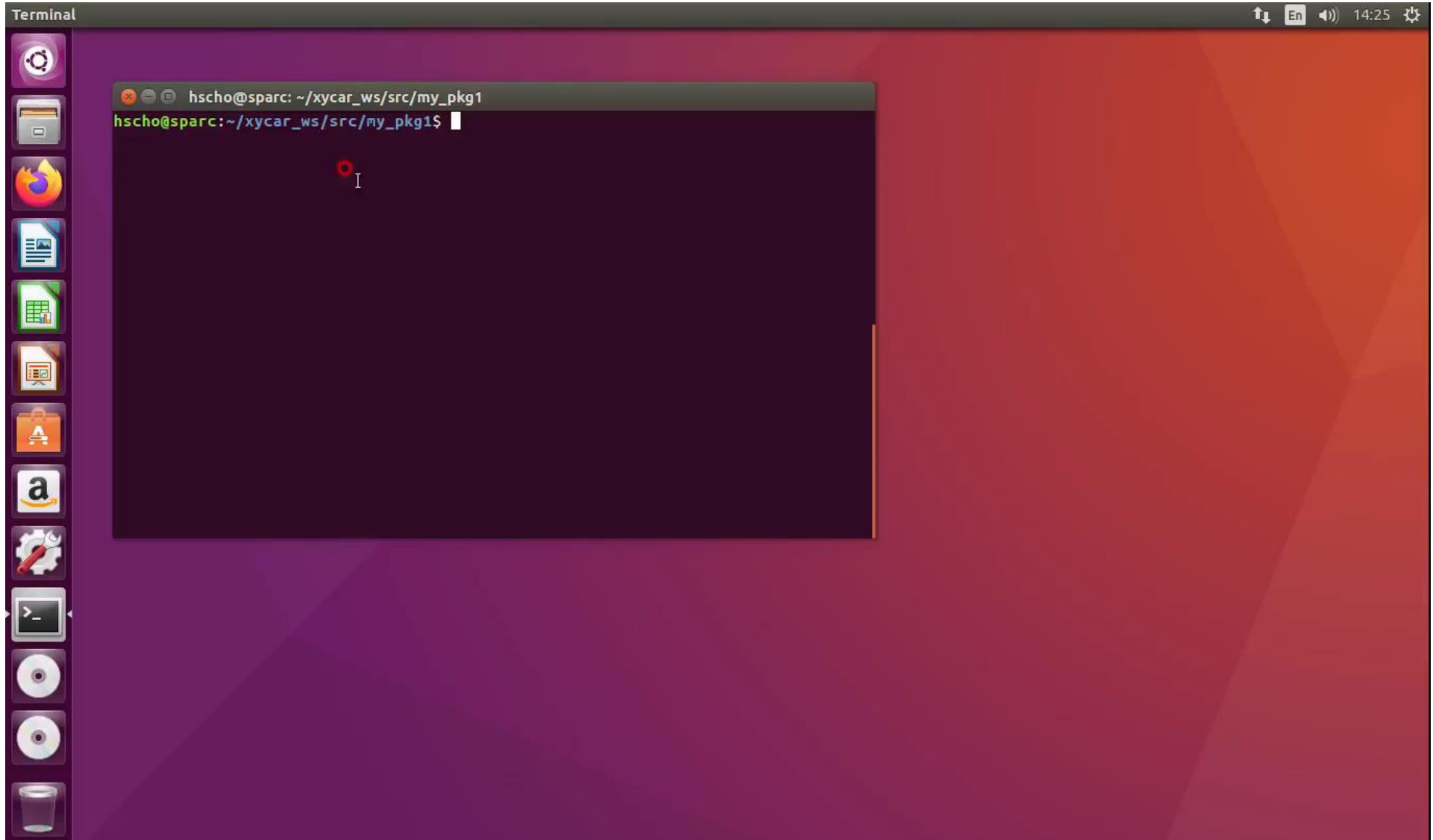


# 노드 동작 확인

- 노드가 잘 동작하고 있는지 확인
  - `$ rqt_graph`

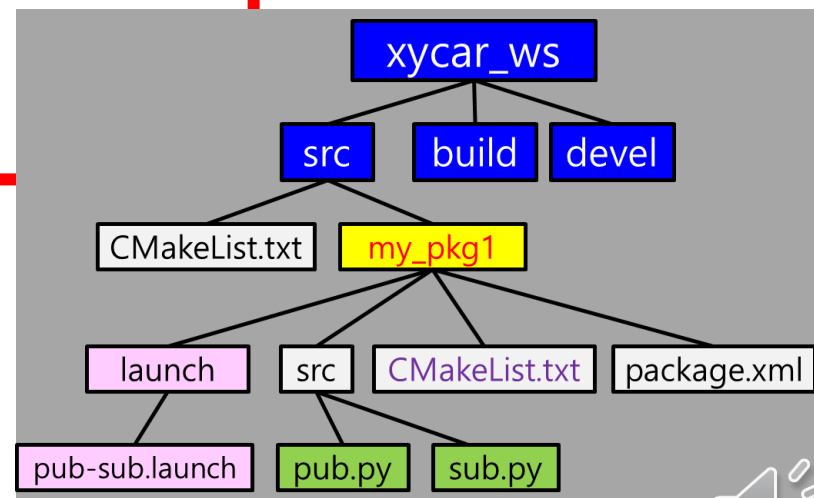


# 실습 (Launch 파일 만들기)



# 정리하면

```
$ cd ~/xycar_ws/src/my_pkg1
$ mkdir launch
$ cd ~/xycar_ws/src/my_pkg1/launch
$ gedit pub-sub.launch
$ cm
$ roslaunch my_pkg1 pub-sub.launch
```





# ROS 프로그래밍 기초

Launch에서 Tag 활용



## \*.launch 파일 사례

- USB 카메라 구동과 파라미터 세팅을 위한 launch 파일
  - 패키지 이름은 'usb\_cam'
  - 파라미터는 5개 (autoexposure, exposure, image\_width, image\_height, camera\_frame\_id)

```
<launch>
  <node name="usb_cam" pkg="usb_cam" type="cam_node" output="screen">
    <param name="autoexposure" value="false"/>
    <param name="exposure" value="150"/>
    <param name="image_width" value="640" />
    <param name="image_height" value="480" />
    <param name="camera_frame_id" value="usb_cam" />
  </node>
</launch>
```



## \*.launch에서 사용하는 Tag : param

- param 태그

- ROS 파라미터 서버에 변수를 등록하고 그 변수에 값을 설정하기 위한 태그

```
<param name="변수의 이름" type="변수의 타입" value="변수 값" />
```

- 속성

- ▶ name : 등록할 변수의 이름
- ▶ type : 등록할 변수의 타입. 사용 할 수 있는 타입의 종류는 str, int, double, bool, yaml
- ▶ value : 등록할 변수의 값

- ROS 파라미터 서버에 등록된 변수는 노드 코드에서 불러와 사용할 수 있음.

```
<node pkg="패키지 명" type="노드가 포함된 소스파일 명" name="노드" output="screen" >
  <param name="age" type="int" value="11"/>
</node>
```

```
import rospy
rospy.init_node('노드')
print(rospy.get_param('~age'))
```

private parameter는  
앞에 '~' 붙인다.



# Launch 파일에서 파라미터 전달 실습

- Launch 파일을 새로 만들어서
  - 파라미터 값을 .lanuch 파일에서 읽어서
  - 그에 맞게 동작하게끔 만들어보자.

새로운 Launch 파일  
pub-sub-param.launch



Launch 파일을 수정하면  
거북이의 회전 반경이  
변경된다.

# Launch 파일에서 파라미터 전달 실습

- Launch 파일을 새로 만든다.
  - \$ gedit pub-sub-param.launch

```
<launch>
  <node pkg="turtlesim" type="turtlesim_node" name="turtlesim_node"/>
  {
    <node pkg="my_pkg1" type="pub_param.py" name="node_param">
      <param name="circle_size" value="2" />
    </node>
    <node pkg="my_pkg1" type="sub.py" name="sub_node" output="screen"/>
  }
</launch>
```

- 패키지 이름 = my\_pkg1
- 타입 (소스코드 파일) = pub\_param.py
- 노드 이름 = node\_param
- 파라미터 이름 = circle\_size
- 파라미터 값 = 2

# Launch 파일에서 파라미터 전달 실습

- pub.py 파일을 복사+수정해서 pub\_param.py 파일을 새로 만든다.
  - \$ cp pub.py pub\_param.py
  - \$ gedit pub\_param.py

```
def move():  
    rospy.init_node('my_node', anonymous=True)  
    pub = rospy.Publisher('/turtle1/cmd_vel', Twist, queue_size=10)  
    msg = Twist()
```

```
# msg.linear.x = 2.0
```

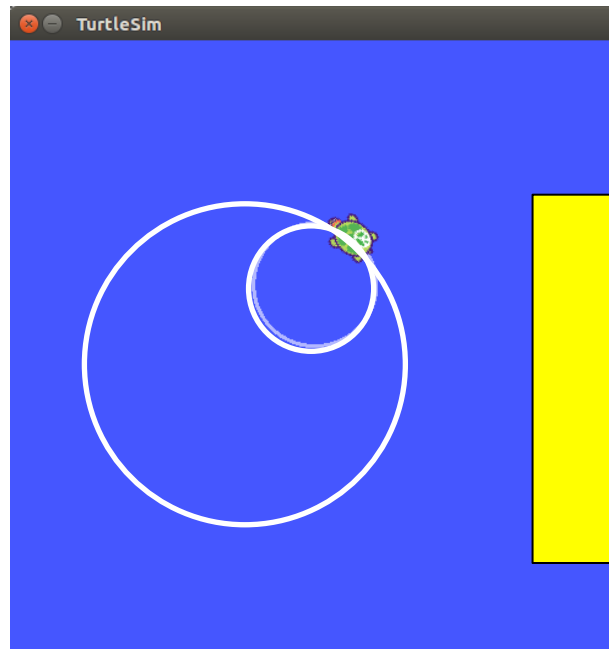
```
linear_x=rospy.get_param('~circle_size')  
msg.linear.x = linear_x
```

```
msg.linear.y = 0.0  
msg.linear.z = 0.0  
msg.angular.x = 0.0  
msg.angular.y = 0.0  
msg.angular.z = 1.8
```

.launch 파일에서  
"circle\_size" param 값을  
읽어들여 사용한다.

# Launch 파일 실행

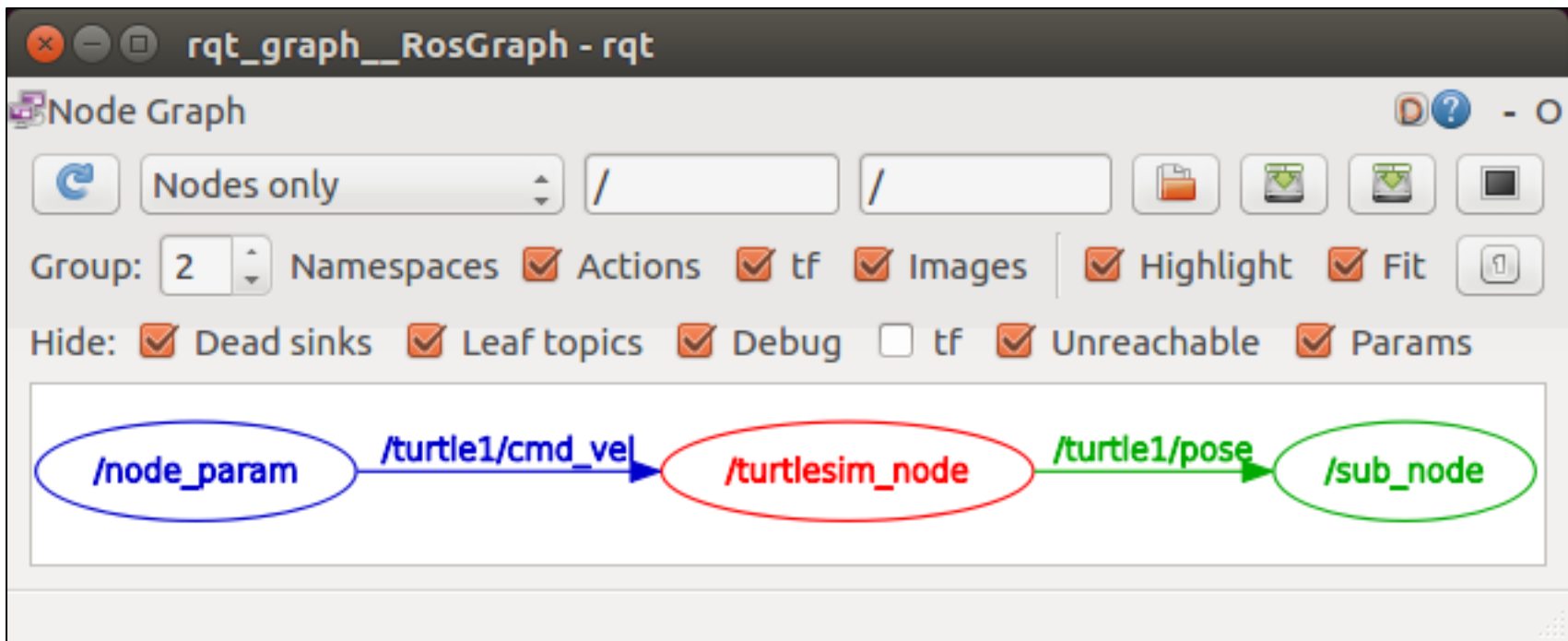
- pub-sub-param.launch 파일에서 values 값을 2 또는 4로 바꿔보자
  - `<param name="circle_size" value="2" />`
- (실행하기) `$ roslaunch my_pkg1 pub-sub-param.launch`



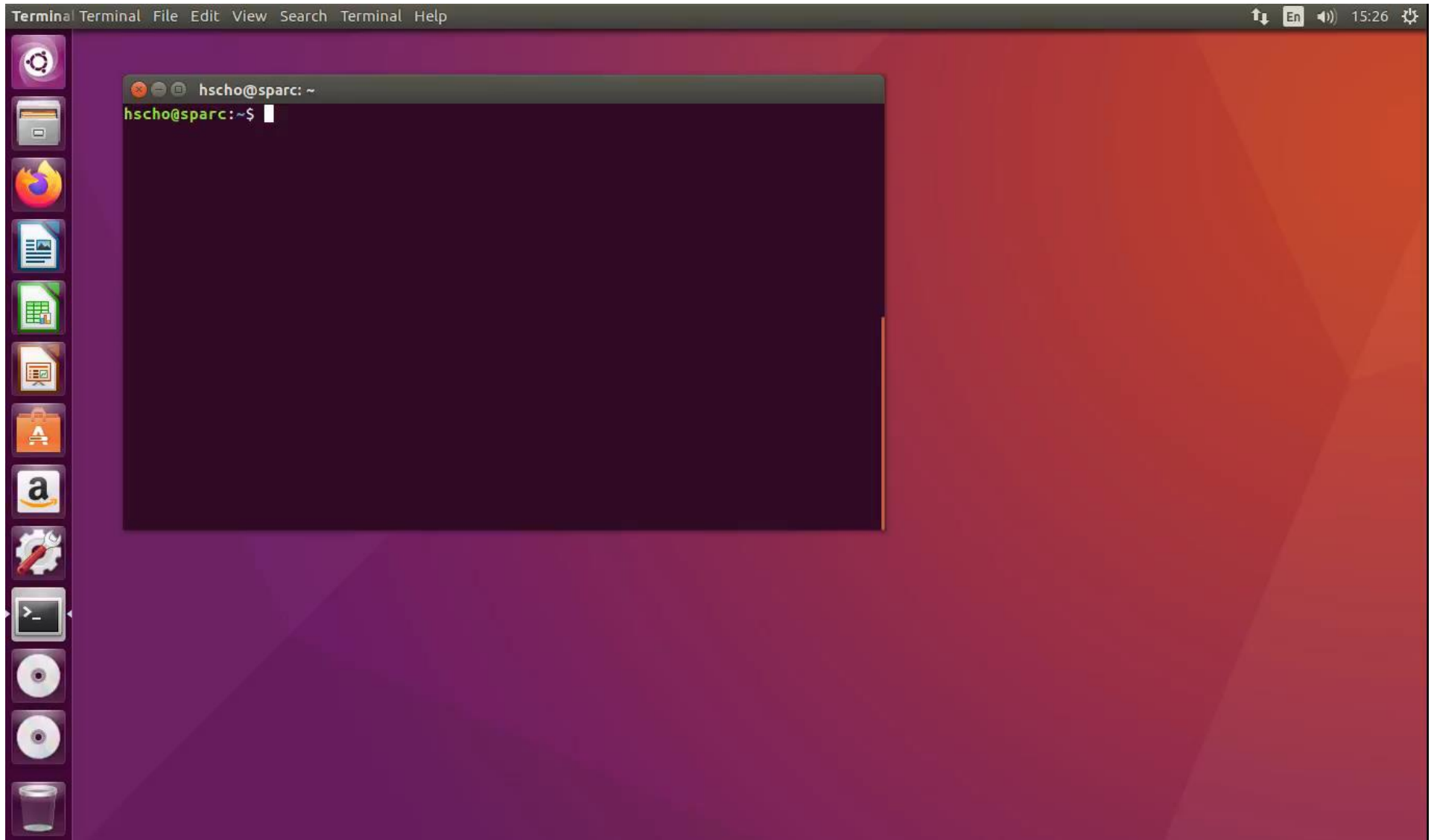
Launch 파일에서  
value 값을 수정하면  
거북이의 회전반경이  
바뀐다.

# 노드 동작 확인

- \$ rqt\_graph
  - “/node\_param” 이름의 노드에서 토픽이 발행됨을 볼 수 있다.



# 실습 (Launch파일 파라미터)





# Q&A

---



감사합니다.



(주)자이트론

허성민

smher@xytron.co.kr

