

LIMO Basics

WeGo & LIMO

목 차

1. Ubuntu
2. Ubuntu 기본 명령어
3. ROS
4. ROS CLI
5. ROS Publisher / Subscriber

01

Ubuntu

01 Ubuntu

- Ubuntu는 Linux 운영체제의 대표적인 배포판 중 하나
- 운영체제 : PC를 쉽게 사용할 수 있게 도와주는 역할 수행



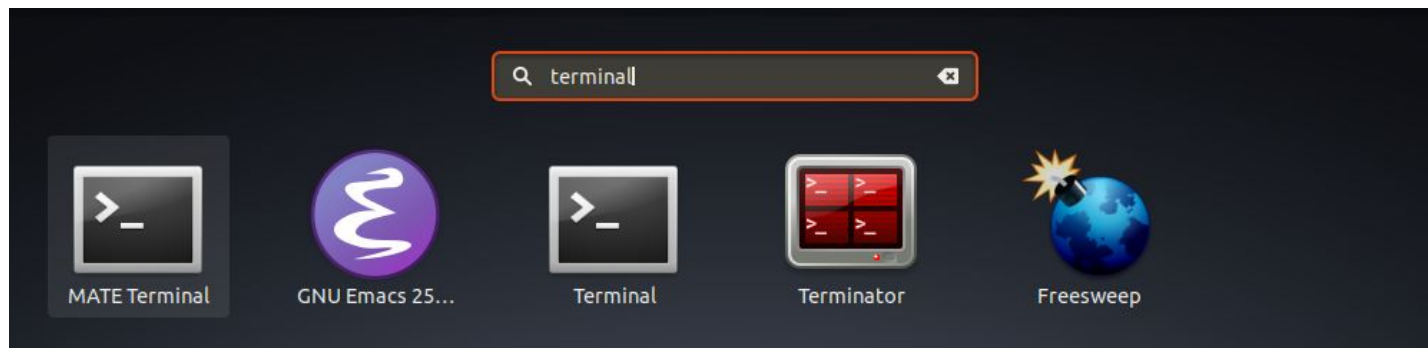
Linux

02

Ubuntu 기본 명령어

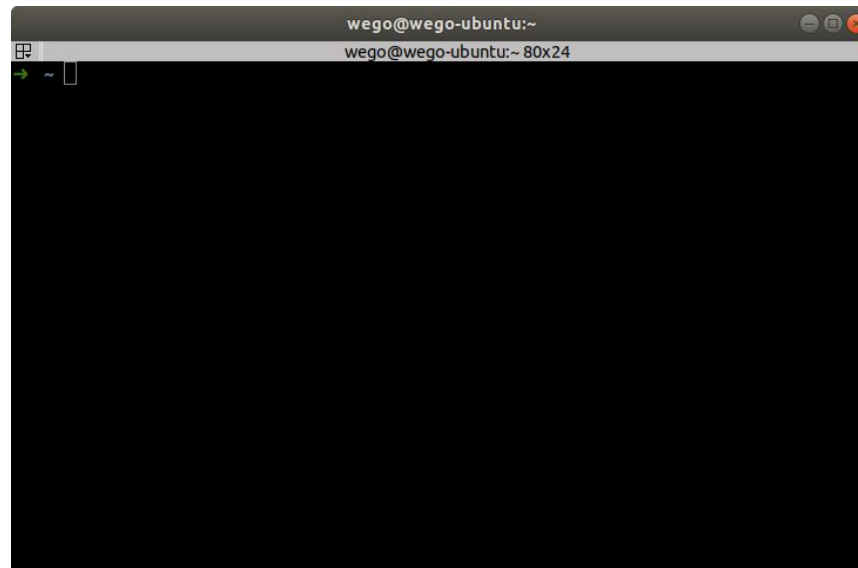
02 Ubuntu 기본 명령어

- Terminal
 - Terminal은 Ubuntu에서 특정 내용의 수행을 위해 명령을 내리는 일종의 단말기 역할을 수행한다.
 - Ctrl + Alt + T 의 단축키를 입력하여, 실행할 수 있다.
 - 또는 windows 버튼을 누르고, 검색 창에 terminal을 입력한 후, 선택하여 실행할 수도 있다.



02 Ubuntu 기본 명령어

- Terminal
 - 아래와 같은 터미널에 입력하여 명령을 내릴 수 있는데 이를 CLI라고 한다.
(Command Line Interface)
 - 이 후 부분에서 해당 터미널에서 사용하는 명령어를 확인해보자.
 - 이 후 부분에서 \$ 뒤에 나오는 명령어는 Terminal에 입력하는 명령어를 의미한다.



02 Ubuntu 기본 명령어

- 패키지 설치하기
 - 일반적인 Windows 환경과 달리, Linux 환경에서는 특정 패키지를 설치할 때 apt를 활용한다. (Advanced Package Tool)
 - `$ sudo apt install <PACKAGE_NAME>`
 - 위 명령어를 통해서 특정 패키지를 설치할 수 있다. (<PACKAGE_NAME> 부분에 설치하려는 패키지를 입력하면 됩니다.)
 - 앞 부분의 sudo 명령어는 sudo 이후에 적힌 내용을 root 권한으로 실행하겠다는 의미이며, 패키지 설치에 필수적으로 sudo를 함께 사용해야 하며, root 권한으로 실행 시 비밀번호를 입력해야 합니다. 기본 비밀번호는 agx 입니다.
(사용하지 않을 경우, 권한 오류가 발생한다.)

02 Ubuntu 기본 명령어

- 패키지 설치하기

```
wego@wego-ubuntu: ~  
wego@wego-ubuntu: ~ 80x24  
wego@wego-ubuntu:~$ gimp  
  
Command 'gimp' not found, but can be installed with:  
  
sudo snap install gimp # version 2.10.28, or  
sudo apt install gimp  
  
See 'snap info gimp' for additional versions.  
  
wego@wego-ubuntu:~$ apt install gimp  
E: Could not open lock file /var/lib/dpkg/lock-frontent - open (13: Permission denied)  
E: Unable to acquire the dpkg frontend lock (/var/lib/dpkg/lock-frontent), are you root?  
wego@wego-ubuntu:~$
```

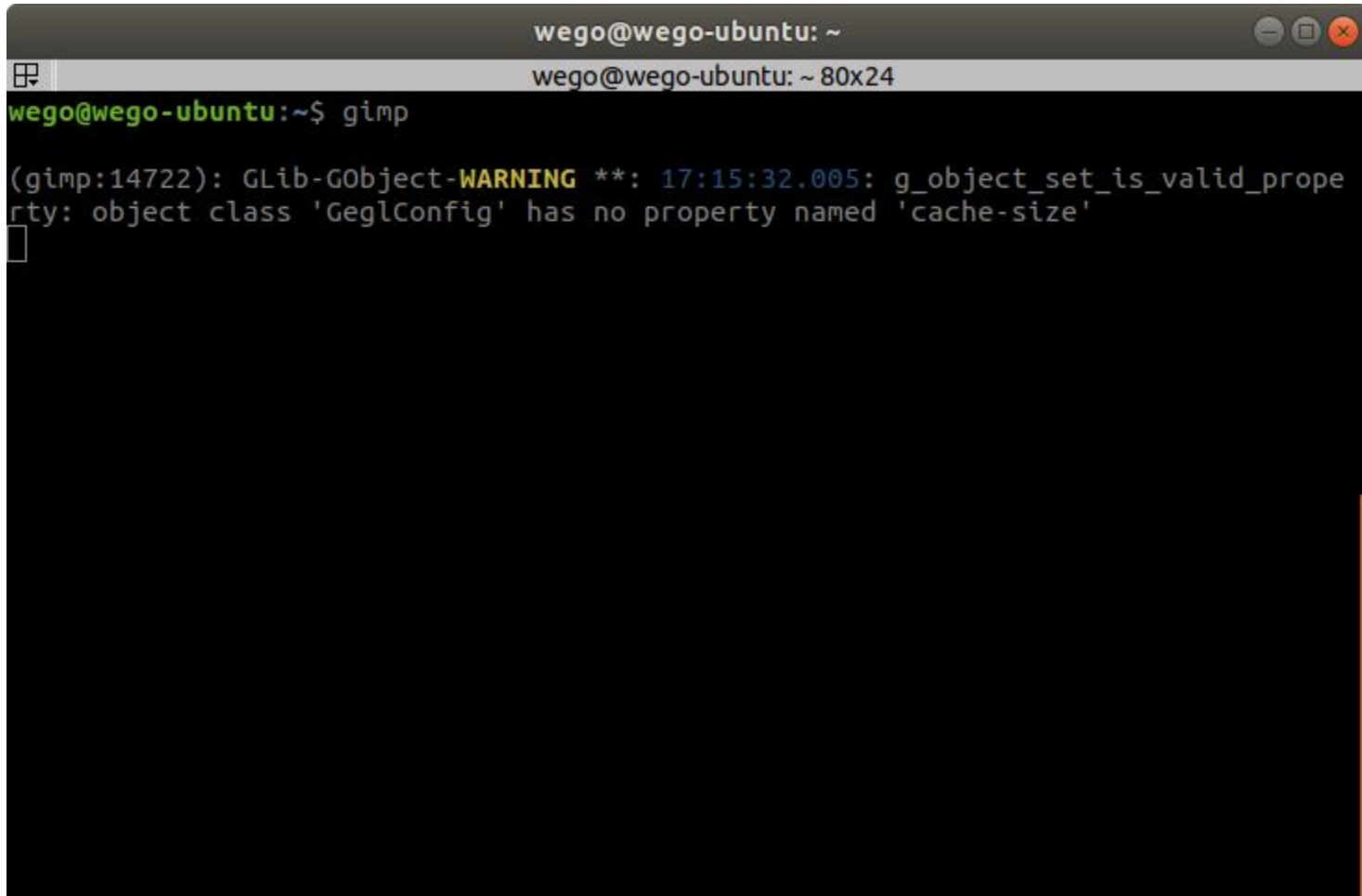
02 Ubuntu 기본 명령어

- 패키지 설치하기

```
wego@wego-ubuntu: ~  
wego@wego-ubuntu: ~ 80x24  
wego@wego-ubuntu:~$ sudo apt install gimp  
[sudo] password for wego:  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
The following additional packages will be installed:  
  gimp-data libbabl-0.1-0 libgegl-0.3-0 libgimp2.0 python-gobject-2  
  python-gtk2  
Suggested packages:  
  gimp-help-en | gimp-help gimp-data-extras python-gobject-2-dbg  
  python-gtk2-doc  
The following NEW packages will be installed:  
  gimp gimp-data libbabl-0.1-0 libgegl-0.3-0 libgimp2.0 python-gobject-2  
  python-gtk2  
0 upgraded, 7 newly installed, 0 to remove and 24 not upgraded.  
Need to get 3,672 kB/14.1 MB of archives.  
After this operation, 74.0 MB of additional disk space will be used.  
Do you want to continue? [Y/n] Y
```

02 Ubuntu 기본 명령어

- 패키지 설치하기

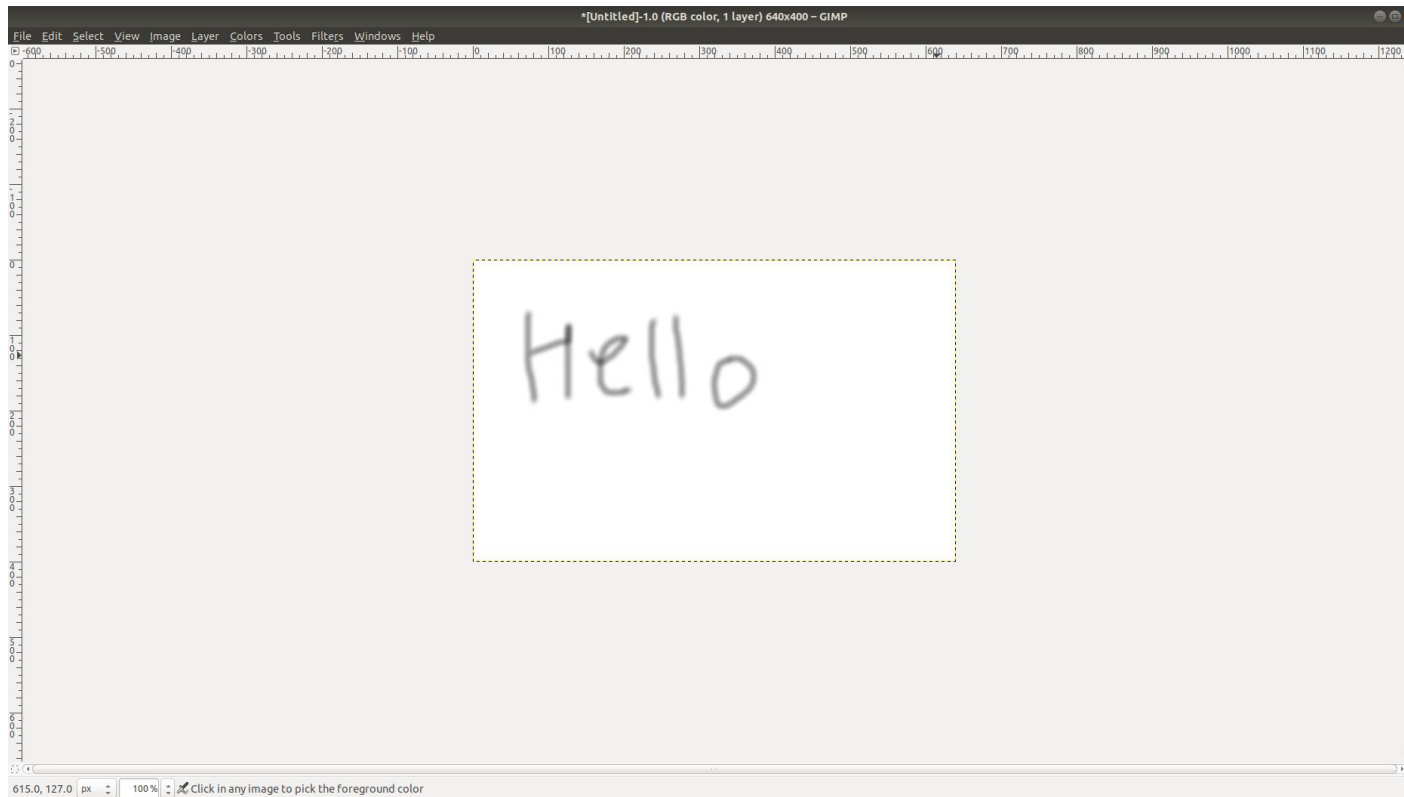


A terminal window titled 'wego@wego-ubuntu: ~' with a standard Ubuntu window header. The terminal shows the command 'gimp' being executed. Below the command, a warning message is displayed: '(gimp:14722): GLib-GObject-WARNING **: 17:15:32.005: g_object_set_is_valid_prope rty: object class 'GeglConfig' has no property named 'cache-size''. The terminal background is black with green text for the prompt and white text for the command and warning.

```
wego@wego-ubuntu: ~  
wego@wego-ubuntu: ~ 80x24  
wego@wego-ubuntu:~$ gimp  
(gimp:14722): GLib-GObject-WARNING **: 17:15:32.005: g_object_set_is_valid_prope  
rty: object class 'GeglConfig' has no property named 'cache-size'
```

02 Ubuntu 기본 명령어

- 패키지 설치하기



02 Ubuntu 기본 명령어

- 특정 폴더로 이동하기
 - Terminal에서 특정 파일에 접근하거나 수정을 하기 위해서 폴더를 이동해야하는 경우가 있습니다.
 - 기본적으로 Terminal을 실행하면, 사용자의 HOME 폴더에서 시작이 됩니다.
(Terminal의 ~이 사용자의 HOME 폴더를 의미합니다.)
 - `$ cd <이동하려는 폴더>`
 - 위 명령어를 통해서 특정 폴더로 이동할 수 있습니다.
 - `$ cd hello`
 - 위 명령어를 통해서 현재 위치에 포함된 hello라는 폴더로 이동할 수 있습니다.

02 Ubuntu 기본 명령어

- 특정 폴더로 이동하기
 - `$ cd ..`
 - 위 명령어를 통해서 상위 폴더로 이동할 수 있습니다.
 - `$ cd`
 - 단순히 위 처럼 cd를 입력하는 것을 통해서는 다시 사용자의 HOME 위치(~)로 돌아올 수 있습니다.
 - 이미 존재하는 Download 폴더로 이동해보겠습니다.
 - `$ cd Downloads`

02 Ubuntu 기본 명령어

- 특정 폴더로 이동하기

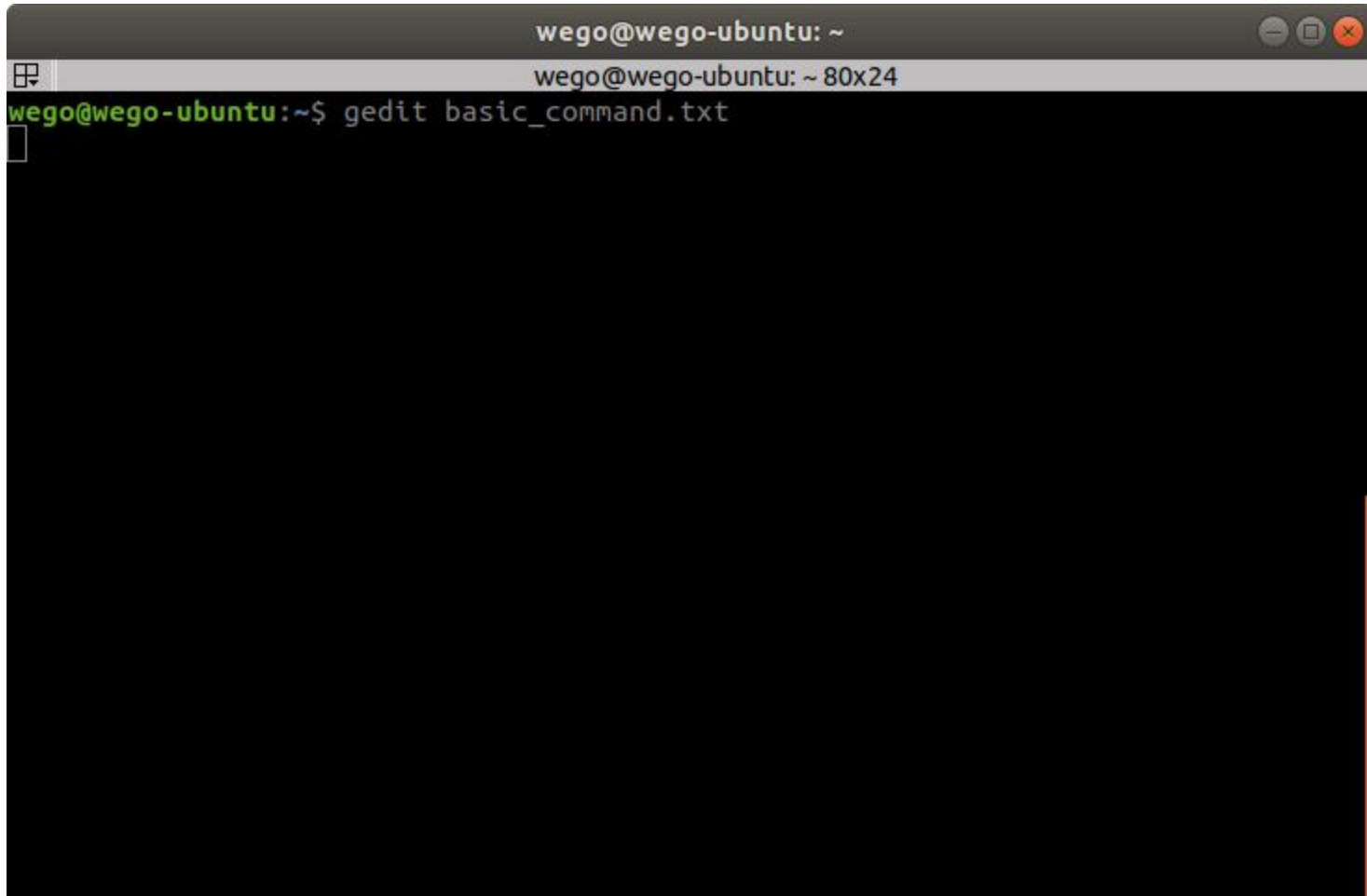
```
wego@wego-ubuntu: ~  
wego@wego-ubuntu: ~ 80x24  
wego@wego-ubuntu:~$ cd Downloads  
wego@wego-ubuntu:~/Downloads$ cd ..  
wego@wego-ubuntu:~$ cd ..  
wego@wego-ubuntu:~/home$ cd  
wego@wego-ubuntu:~$
```

02 Ubuntu 기본 명령어

- 새로운 파일 만들어서 내용 적기
 - Terminal 내부에서 원하는 폴더로 이동했다면, 특정 파일을 만들어서 수정하고 저장할 수 있습니다.
 - 간단하게 gedit을 활용해보겠습니다.
 - `$ gedit basic_command.txt`
 - 위 명령어를 입력하면 basic_command.txt 파일이 생성되고, 이 파일을 텍스트 에디터인 gedit으로 열게 됩니다.
 - 열린 텍스트 에디터에 내용을 적고 Ctrl + S 를 입력하여 저장할 수 있습니다.

02 Ubuntu 기본 명령어

- 새로운 파일 만들어서 내용 적기

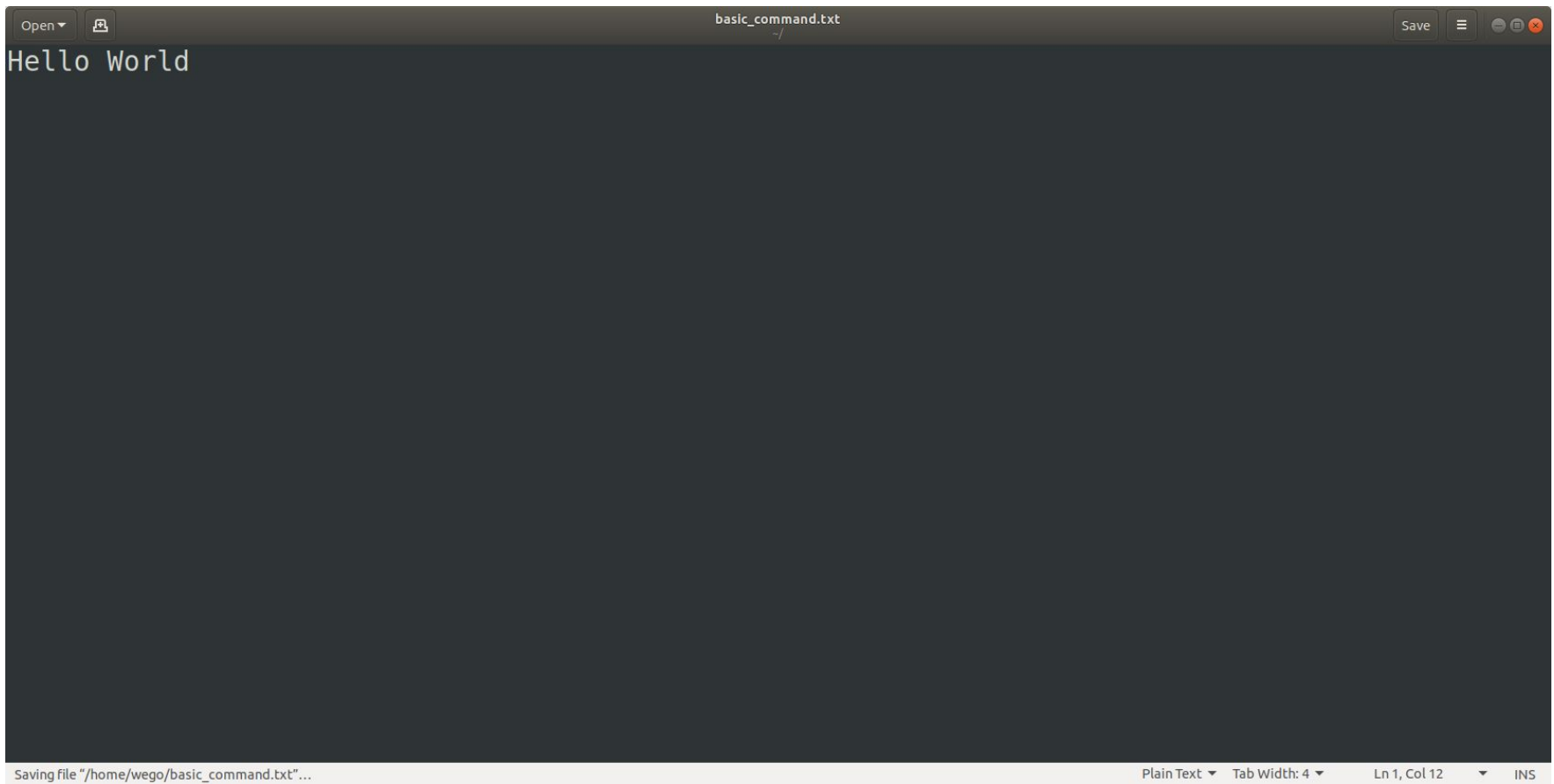


A terminal window titled 'wego@wego-ubuntu: ~' with standard window controls. The terminal shows the command 'wego@wego-ubuntu:~\$ gedit basic_command.txt' entered. The window title bar also displays 'wego@wego-ubuntu: ~ 80x24'.

```
wego@wego-ubuntu: ~  
wego@wego-ubuntu:~$ gedit basic_command.txt
```

02 Ubuntu 기본 명령어

- 새로운 파일 만들어서 내용 적기



The screenshot shows a text editor window with a dark theme. The title bar at the top indicates the file is named 'basic_command.txt' and is located in the home directory ('~/'). The editor area contains the text 'Hello World' on the first line. The status bar at the bottom shows the file is being saved to '/home/wego/basic_command.txt...', the text is in 'Plain Text' format, the tab width is 4, and the cursor is at line 1, column 12.

```
basic_command.txt
~/
Hello World

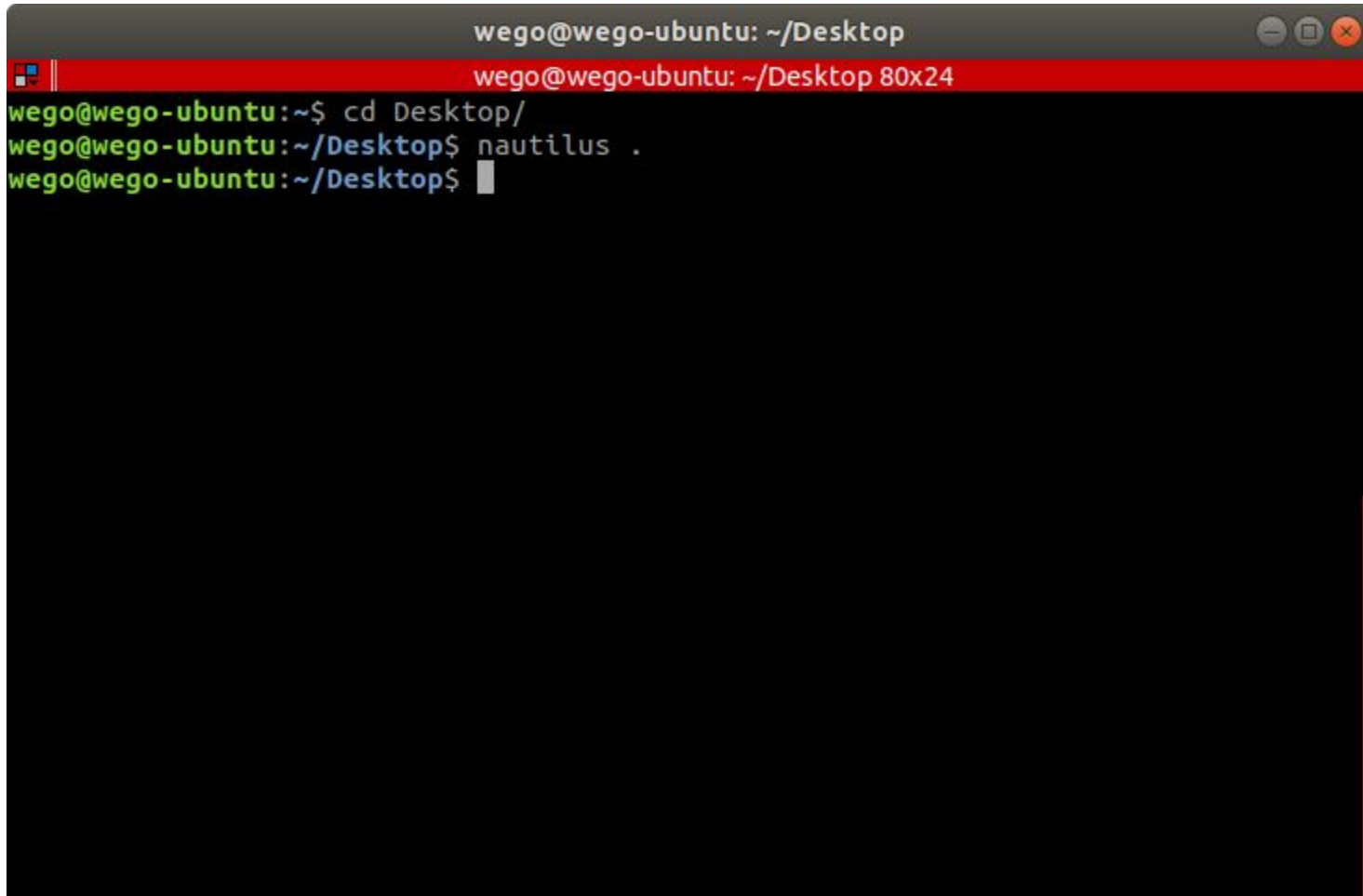
Saving file "/home/wego/basic_command.txt"...
Plain Text Tab Width: 4 Ln 1, Col 12 INS
```

02 Ubuntu 기본 명령어

- 해당 위치에서 파일 탐색기 열기
 - Ubuntu에서도 Windows와 유사하게 파일 탐색기를 사용할 수 있습니다.
 - Ubuntu에서 사용하는 파일 탐색기의 이름은 nautilus 입니다.
 - cd 명령어를 이용하여 원하는 폴더로 이동한 후 아래 명령어를 통해서 파일 탐색기를 열 수 있습니다.
 - `$ nautilus .`
 - 파일 탐색기에서 폴더 만들기, 다른 폴더의 이동, 폴더 내부 파일 확인 등의 다양한 기능을 확인할 수 있습니다.

02 Ubuntu 기본 명령어

- 해당 위치에서 파일 탐색기 열기

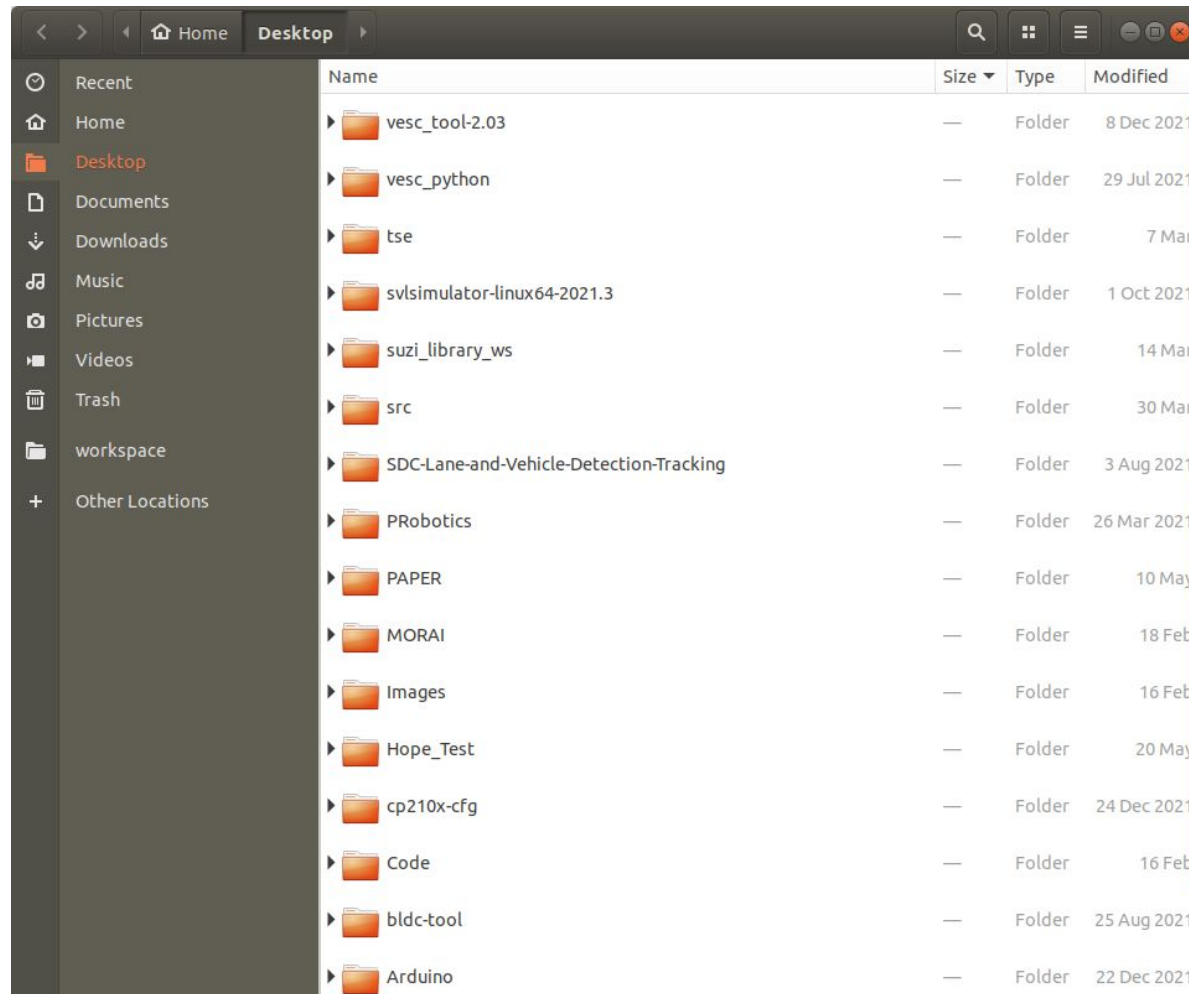


```
wego@wego-ubuntu: ~/Desktop
wego@wego-ubuntu: ~/Desktop 80x24
wego@wego-ubuntu:~$ cd Desktop/
wego@wego-ubuntu:~/Desktop$ nautilus .
wego@wego-ubuntu:~/Desktop$
```

02 Ubuntu 기본 명령어

- 해당 위치에서 파일 탐색기 열기

•



02 Ubuntu 기본 명령어

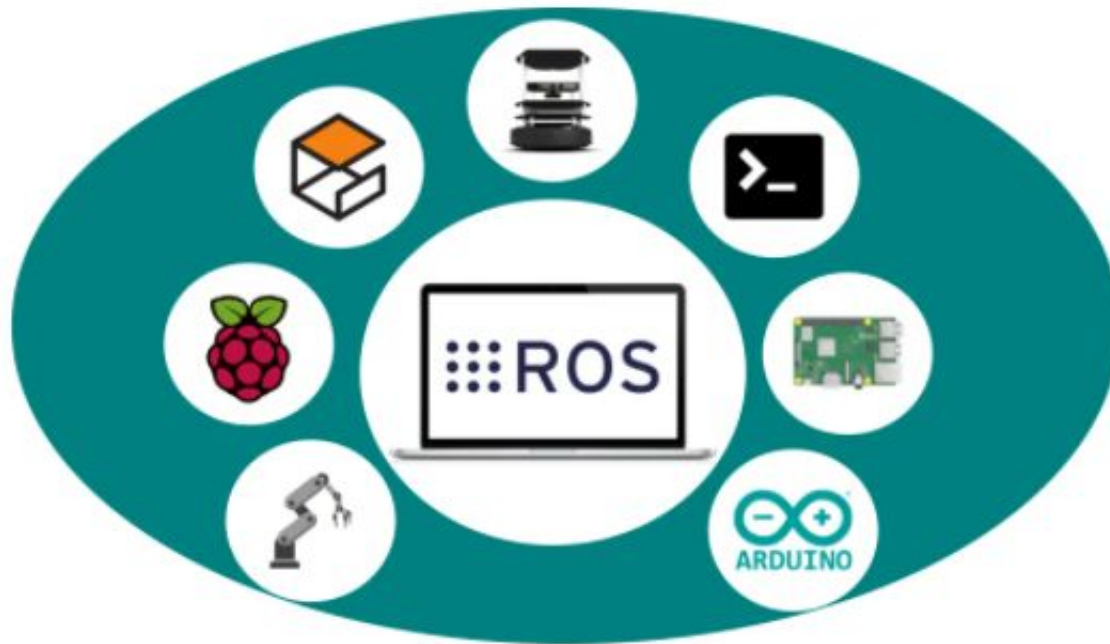
- 더 많은 명령어 및 사용 방법은
- <https://ubuntu.com/tutorials/command-line-for-beginners#1-overview>
- 위 사이트를 통해서 확인할 수 있습니다.

03

ROS

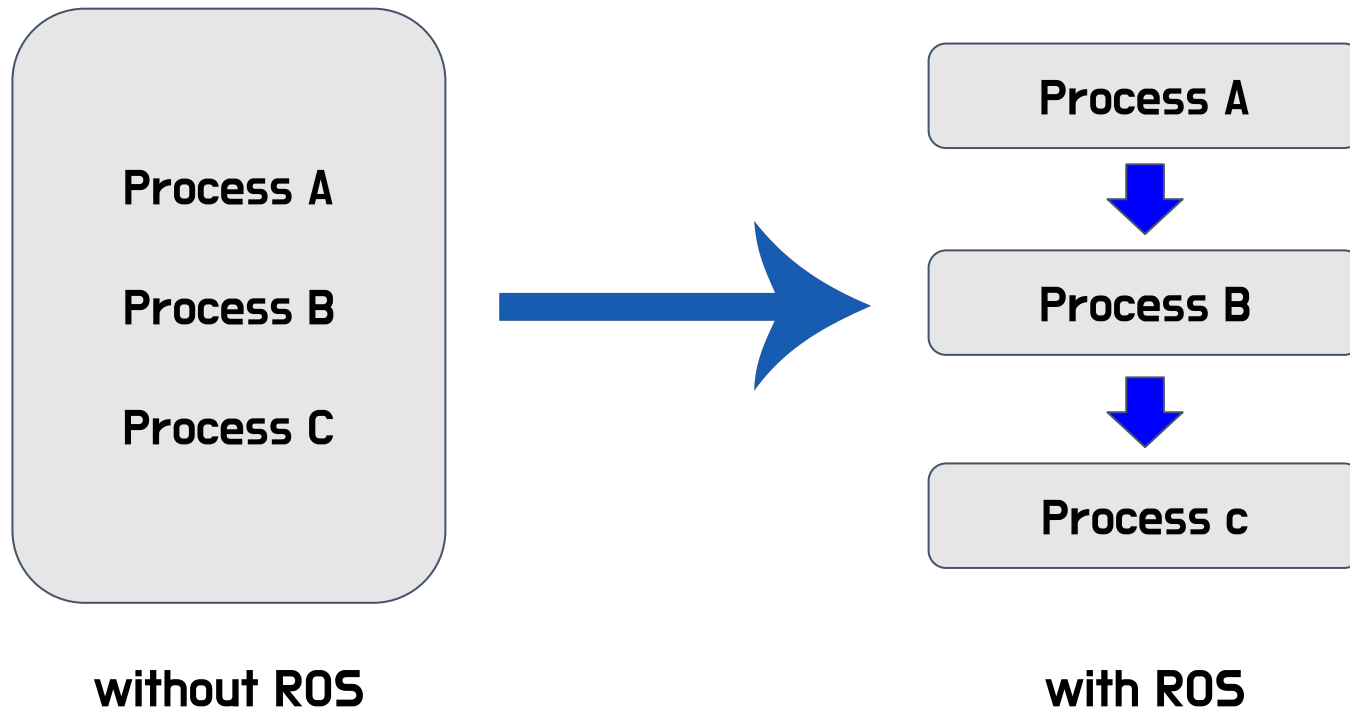
03 ROS

- ROS는 로봇을 제어하기 위해 사용하는 중간 다리 역할을 수행합니다.
- Ubuntu와 로봇을 연결해주고, 데이터를 주고 받을 수 있게 해줍니다.



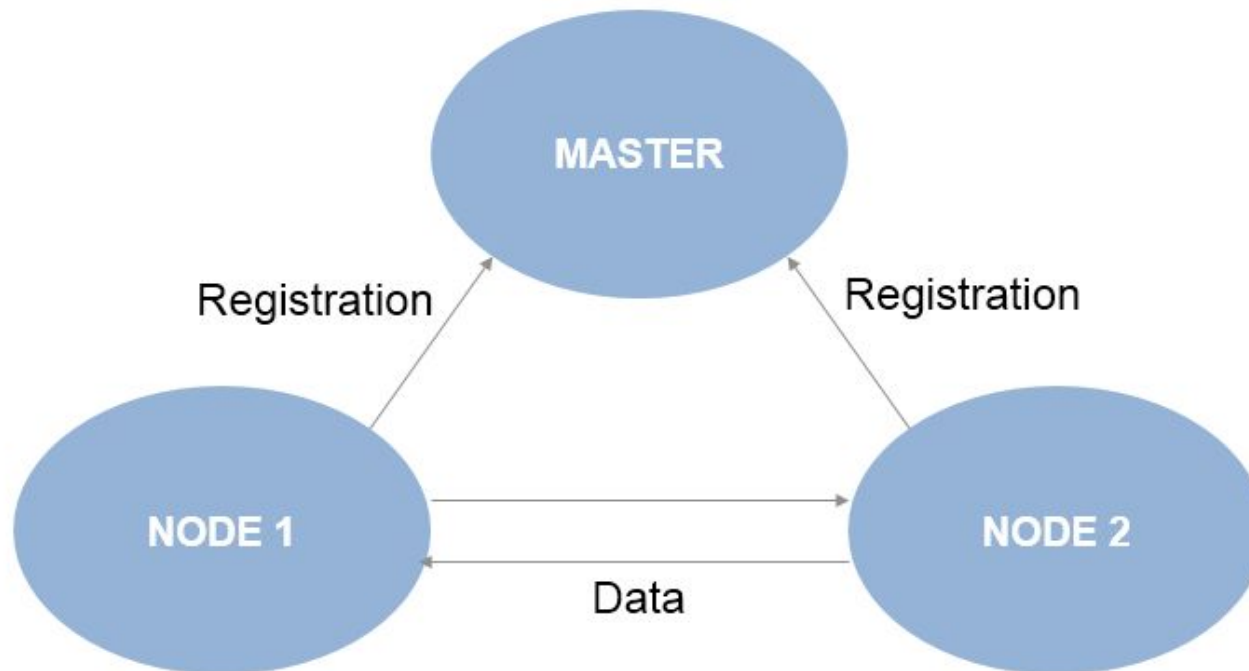
03 ROS

- ROS는 대표적으로 C++와 Python을 이용하여 쉽게 사용할 수 있습니다.
- 또한 ROS를 통해 여러 개의 실행 파일 사이의 통신을 쉽게 할 수 있습니다.



03 ROS

- ROS는 Master와 이에 연결되는 Node로 이루어져 있으며, Master가 Node를 관리하고, 통신을 도와주는 역할을 수행한다.
- ROS Master가 없을 경우, Node 사이의 통신이 추가적으로 불가능해진다.
- 새로운 Node의 실행 또한 불가능해집니다.



04

ROS CLI

04 ROS CLI

- ROS를 설치하고 나면, Terminal에서 ROS에 관련된 명령어도 사용을 할 수 있게 됩니다.
- Terminal에서 사용 가능한 ROS 관련 명령어를 설명하겠습니다.
- 대표적인 명령어는 다음과 같습니다
- `roscore`, `roslaunch`, `rostopic`, `rosbag`, `rosmmsg`,
`catkin_create_pkg`, `catkin_make`, `rosdep`, `rqt`, `rviz`

04 ROS CLI

- ROS Master를 실행하기
- `$ roscore`
- 위 명령어를 수행하면, ROS Master를 실행할 수 있습니다.

04 ROS CLI

- ROS Master를 실행하기

```
roscore http://wego-ubuntu:11311/
roscore http://wego-ubuntu:11311/80x31
wego@wego-ubuntu:~$ roscore
... logging to /home/wego/.ros/log/16164f02-e63f-11ec-b10a-d83bbf196793/roslaunc
h-wego-ubuntu-18651.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://wego-ubuntu:45747/
ros_comm version 1.14.13

SUMMARY
=====

PARAMETERS
* /rostdistro: melodic
* /rosversion: 1.14.13

NODES

auto-starting new master
process[master]: started with pid [18672]
ROS_MASTER_URI=http://wego-ubuntu:11311/

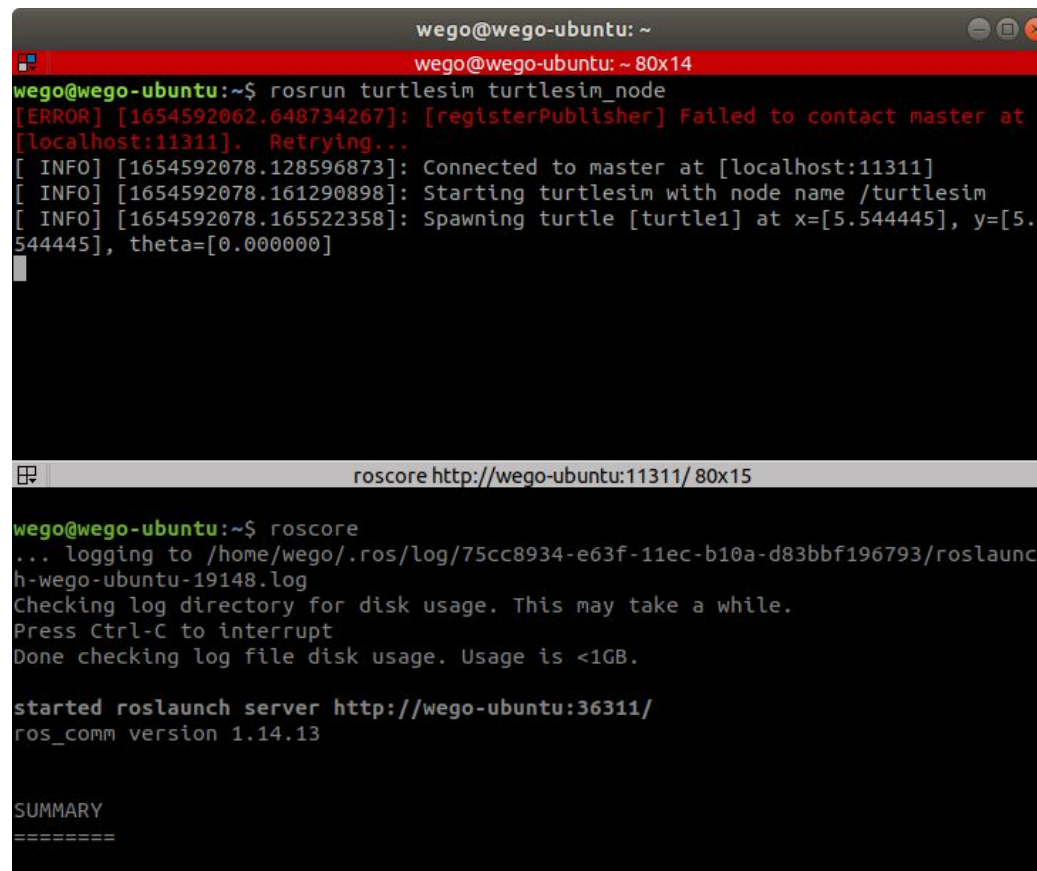
setting /run_id to 16164f02-e63f-11ec-b10a-d83bbf196793
process[rosout-1]: started with pid [18694]
started core service [/rosout]
```

04 ROS CLI

- ROS Node를 실행하기
- 가장 기본적으로 실행할 수 있는 ROS 파일을 ROS Node라고 합니다.
- 이러한 ROS Node는 Master가 존재하고 있는 상태에서만 실행할 수 있습니다.
- `$ rosrun <PACKAGE_NAME> <NODE_NAME>`
- <PACKAGE_NAME> 부분에 실행할 패키지의 이름을 적습니다.
- <NODE_NAME> 부분에 해당 패키지 내부의 실행 Node 이름을 적습니다.
- ex) `$ rosrun turtlesim turtlesim_node`

04 ROS CLI

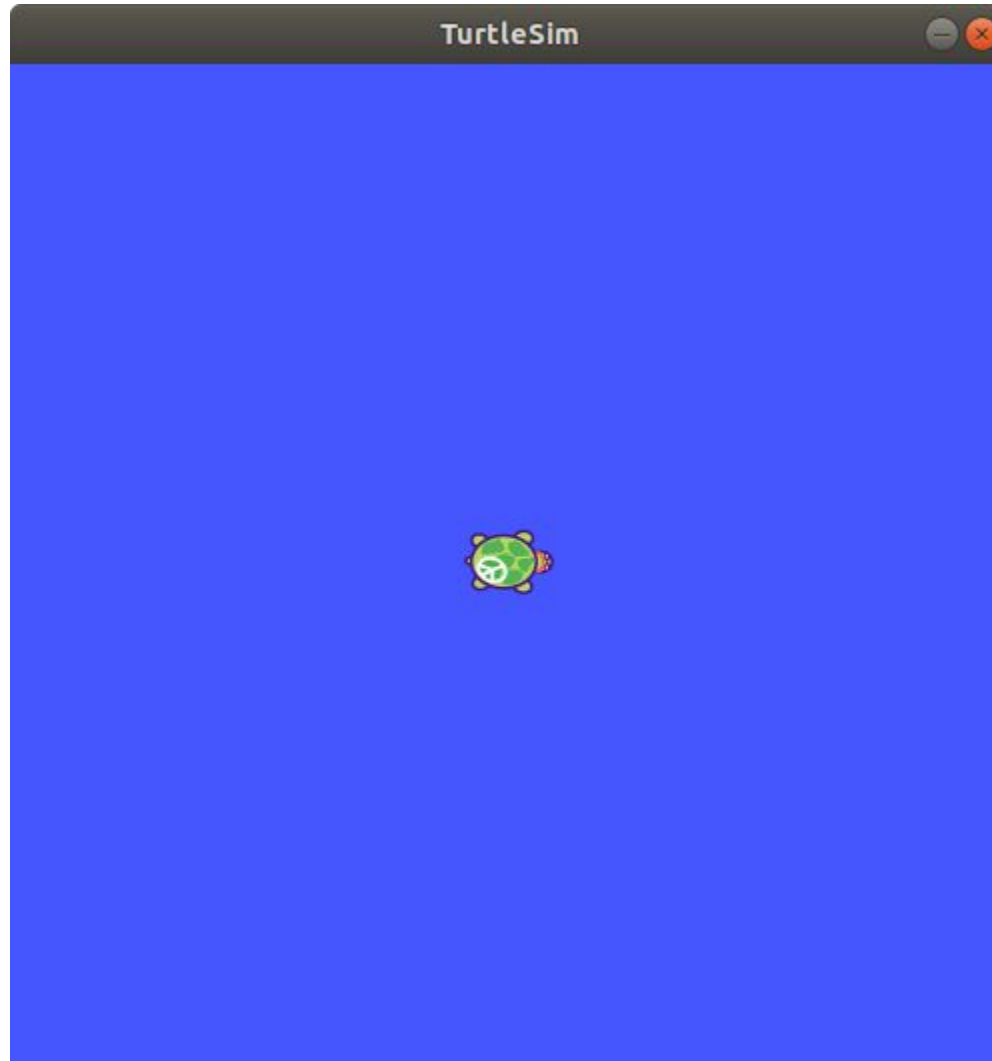
- ROS Node를 실행하기
 - ROS Master가 없을 경우, Failed to contact master at ... 오류가 나며, Master가 있을 때까지, 대기합니다.



```
wego@wego-ubuntu: ~  
wego@wego-ubuntu: ~ 80x14  
wego@wego-ubuntu:~$ rosrn turtlesim turtlesim_node  
[ERROR] [1654592062.648734267]: [registerPublisher] Failed to contact master at  
[localhost:11311]. Retrying...  
[ INFO] [1654592078.128596873]: Connected to master at [localhost:11311]  
[ INFO] [1654592078.161290898]: Starting turtlesim with node name /turtlesim  
[ INFO] [1654592078.165522358]: Spawning turtle [turtle1] at x=[5.544445], y=[5.  
544445], theta=[0.000000]  
[roscore http://wego-ubuntu:11311/ 80x15]  
wego@wego-ubuntu:~$ roscore  
... logging to /home/wego/.ros/log/75cc8934-e63f-11ec-b10a-d83bbf196793/roslaunc  
h-wego-ubuntu-19148.log  
Checking log directory for disk usage. This may take a while.  
Press Ctrl-C to interrupt  
Done checking log file disk usage. Usage is <1GB.  
  
started roslaunch server http://wego-ubuntu:36311/  
ros_comm version 1.14.13  
  
SUMMARY  
=====
```


04 ROS CLI

- ROS Node를 실행하기



04 ROS CLI

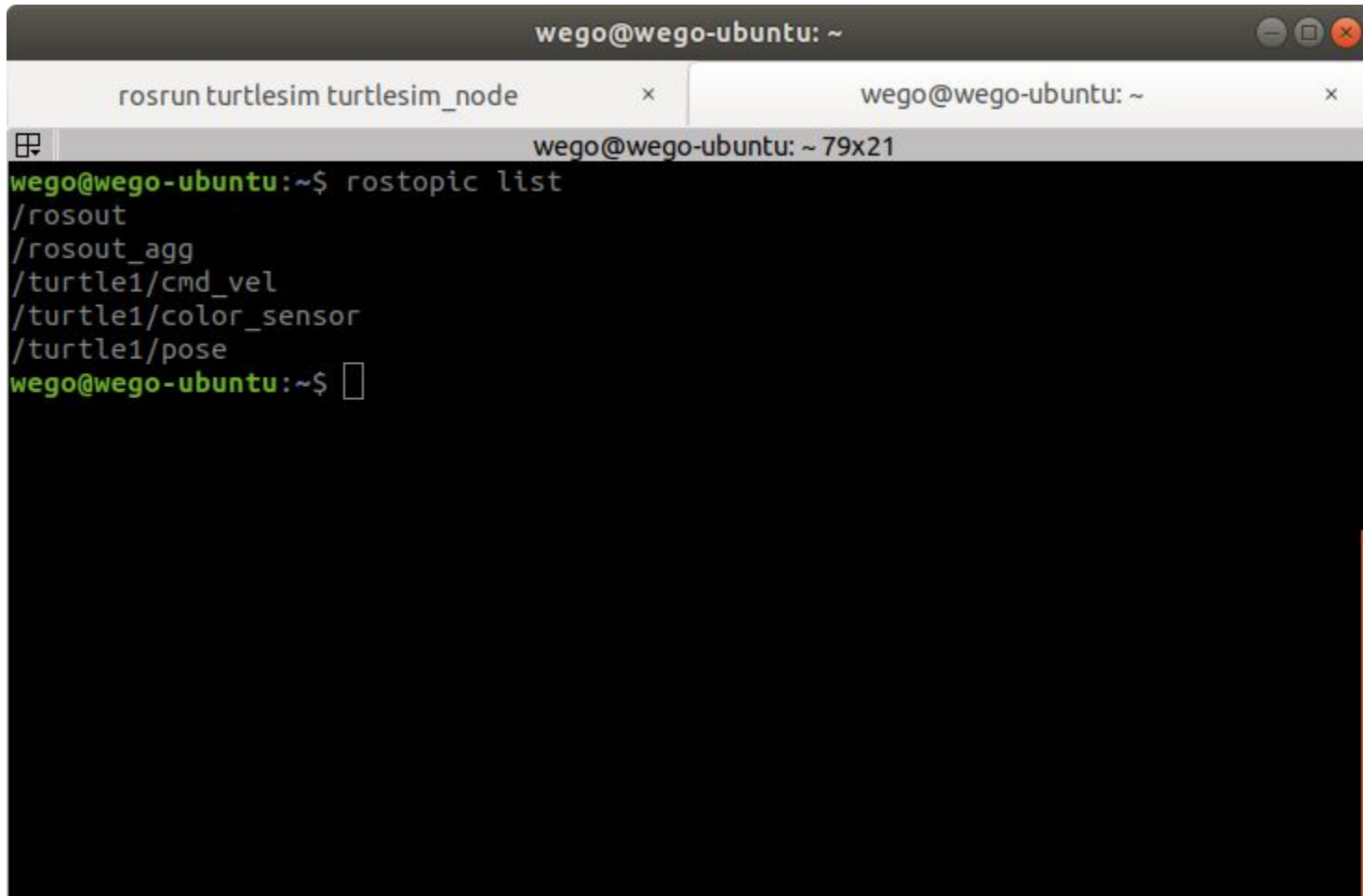
- ROS Node 여러 개 실행하기
- roslaunch 명령어를 통해, 여러 개의 Node를 한 번에 실행할 수 있습니다.
- rosrun 명령어와 다르게, roslaunch는 Master가 없을 경우, Master를 실행한 후, 해당 Node들을 실행합니다.
- `$ roslaunch <PACKAGE_NAME> <LAUNCH_FILE>`
- <PACKAGE_NAME> 부분에 실행할 launch 파일이 존재하는 패키지의 이름을 적습니다.
- <LAUNCH_FILE> 부분에 실행한 launch파일의 이름을 적습니다.

04 ROS CLI

- ROS Topic에 대한 정보 확인하기
- rostopic 명령어를 통해, topic에 대한 정보를 확인할 수 있습니다.
- Master가 실행된 상태에서, rostopic에 대한 정보를 받아오며, 해당 Topic을 받는 Node, 보내는 Node, 데이터 타입 등에 대한 정보를 확인할 수 있습니다.
- `$ rostopic list`
- 위 명령을 통해, 현재 Publish or Subscribe 되고 있는 Topic을 확인할 수 있습니다.

04 ROS CLI

- ROS Topic에 대한 정보 확인하기



A terminal window titled 'wego@wego-ubuntu: ~' with two tabs: 'roslaunch turtlesim turtlesim_node' and 'wego@wego-ubuntu: ~'. The active tab shows the command 'rostopic list' and its output:

```
wego@wego-ubuntu:~$ rostopic list
/rosout
/rosout_agg
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
wego@wego-ubuntu:~$
```

04 ROS CLI

- ROS Topic에 대한 정보 확인하기
- `$ rostopic echo <Topic Name>`
- 위 명령어를 통해, 해당 Topic에 전달되고 있는 값을 확인할 수 있습니다.

```
wego@wego-ubuntu: ~  
roslaunch turtlesim turtlesim_node x wego@wego-ubuntu: ~  
wego@wego-ubuntu: ~ 79x21  
wego@wego-ubuntu:~$ rostopic list  
/rosout  
/rosout_agg  
/turtle1/cmd_vel  
/turtle1/color_sensor  
/turtle1/pose  
wego@wego-ubuntu:~$ rostopic echo /turtle1/pose -n 1  
x: 5.544444561  
y: 5.544444561  
theta: 0.0  
linear_velocity: 0.0  
angular_velocity: 0.0  
---  
wego@wego-ubuntu:~$
```

04 ROS CLI

- ROS Topic에 대한 정보 확인하기
- `$ rostopic info <Topic Name>`
- 위 명령어를 통해, 해당 Topic에 대한 Type, Publish, Subscribe에 대한 정보를 확인할 수 있습니다.

```
wego@wego-ubuntu: ~ 79x21
wego@wego-ubuntu:~$ rostopic info /turtle1/pose
Type: turtlesim/Pose

Publishers:
* /turtlesim (http://wego-ubuntu:42175/)

Subscribers: None

wego@wego-ubuntu:~$
```

04 ROS CLI

- ROS Topic에 대한 정보 확인하기
- `$ rostopic pub -r <RATE> <Topic Name> <Message Type> <보낼 데이터>`
- 위 명령어를 통해, 특정 Topic에 데이터를 보낼 수 있습니다.

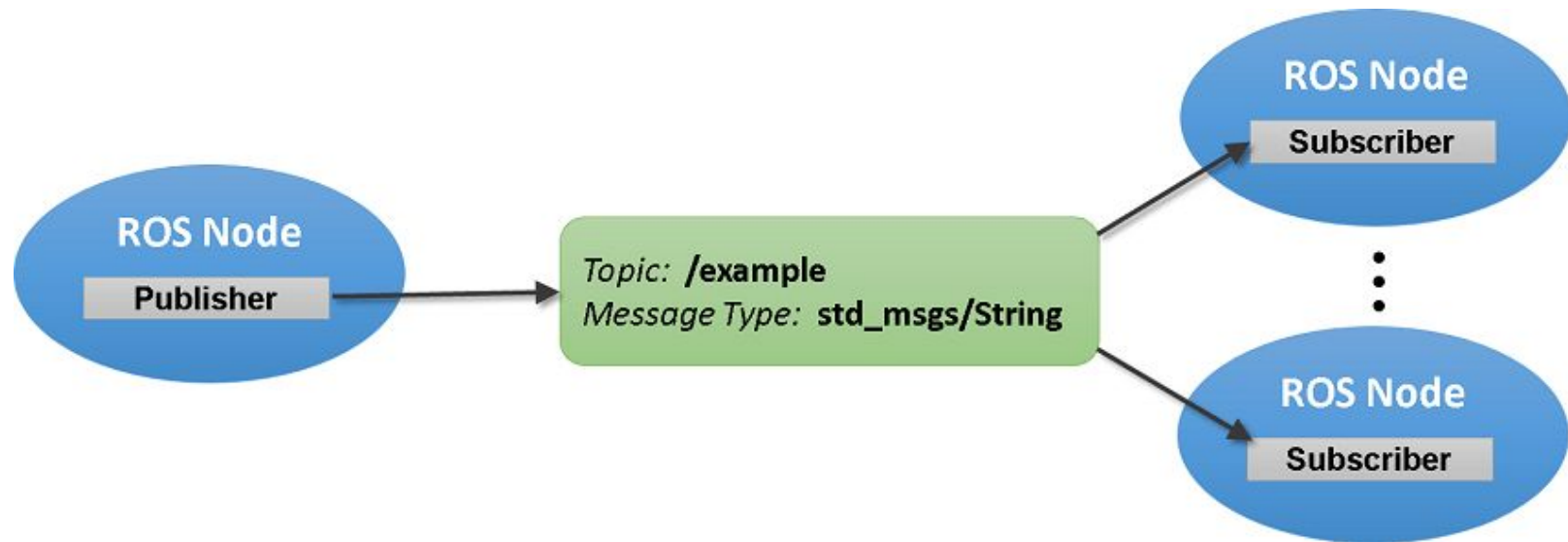
```
wego@wego-ubuntu: ~ 79x21
wego@wego-ubuntu:~$ rostopic pub -r 5 /turtle1/cmd_vel geometry_msgs/Twist "linear:
  x: 0.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 1.0"
```

05

ROS Publisher / Subscriber

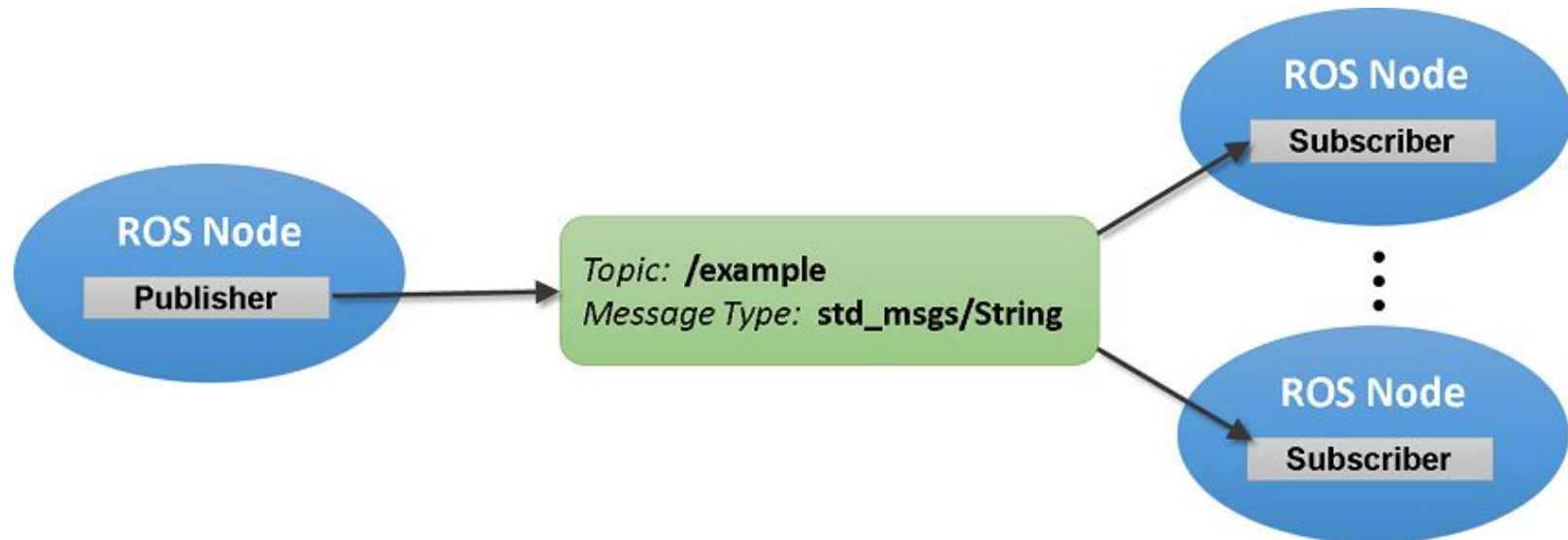
05 ROS Publisher / Subscriber

- ROS 통신 방식
 - 대표적인 통신 방식 중 하나인 ROS Topic
 - ROS Topic은 단방향 통신이고, 지속적으로 데이터를 송수신할 수 있습니다.



05 ROS Publisher / Subscriber

- ROS 통신 방식
 - 보내는 쪽을 Publisher Node, 받는 쪽을 Subscriber Node라고 합니다.
 - `$ rostopic echo` 명령을 통해, 간단한 Topic Subscriber를
 - `$ rostopic pub` 명령을 통해, 간단한 Publisher를 수행할 수 있습니다.
 - Python 기반의 간단한 Publisher 및 Subscriber를 만들어보겠습니다.



05 ROS Publisher / Subscriber

- Prerequisites
 - 우선 해당 내용을 수행할 Workspace를 생성해보겠습니다.
 - 아래 명령을 입력하여, 새로운 Workspace를 생성하고, 사용할 준비를 합니다.
 - `$ cd ~ && mkdir -p catkin_ws/src && cd catkin_ws/src`
 - `$ catkin_create_pkg ros_tutorial rospy`
 - `$ mkdir -p ros_tutorial/scripts`
 - `$ cd ../.. && catkin_make`
 - `$ source devel/setup.bash`

05 ROS Publisher / Subscriber

- Prerequisites
 - 이 후의 내용은 생성한 catkin_ws를 워크스페이스로 사용합니다.
 - 생성하는 파일들을 ~/catkin_ws/src/ros_tutorial/scripts/ 내부에 위치해야 합니다.
 - Python으로 최종 파일을 생성한다면 위치는
~/catkin_ws/src/ros_tutorial/scripts/test_script.py 입니다.

05 ROS Publisher / Subscriber

- ROS Publisher
 - 우선 Publisher 생성을 위해 turtle_drive_pub.py 파일을 생성합니다.
 - 최상단에는 ROS가 어떤 인터프리터로 실행될지 지정해주어야하기 때문에 이에 해당하는 shebang도 있어야합니다. 이에 대한 내용을 작성하면 다음과 같습니다.
 - ROS에서 동작하는 Node를 생성하기 위해 rospy 모듈을 import해야합니다.
 - 또한 송수신할 데이터 타입을 import해야합니다. 여기서는 geometry_msgs/Twist을 사용하겠습니다 (Twist 사용을 위한 Vector3 포함)

```
#!/usr/bin/env python
import rospy
from geometry_msgs.msg import Twist, Vector3
```

05 ROS Publisher / Subscriber

- ROS Publisher
 - 다음으로 필요한 내용은 ROS에 해당 Node를 등록하는 부분입니다.
 - rospy 내부에 포함된 init_node 메서드를 이용하여 등록할 수 있습니다.
 - init_node 메서드는 등록할 Node의 이름을 필수적으로 명시해주어야합니다.
 - 파일 이름과 동일하게 “turtle_drive_pub”으로 정하겠습니다.
 - 코드는 아래와 같습니다.

```
rospy.init_node("turtle_drive_pub")
```

05 ROS Publisher / Subscriber

- ROS Publisher
 - 다음으로는 데이터 송신을 위한 Publisher 객체를 생성합니다.
 - Publisher 객체는 생성자로 Publish할 Topic의 이름, Topic의 메시지 타입, Queue의 크기를 입력 받습니다.
 - Topic의 이름은 “cmd_vel”, 메시지 타입은 Twist, Queue의 크기는 적당히 10으로 정하겠습니다. 코드는 아래와 같습니다.

```
drive_pub = rospy.Publisher("cmd_vel", Twist, queue_size=10)
```

05 ROS Publisher / Subscriber

- ROS Publisher
 - 다음으로, 데이터를 보낼 속도를 지정하겠습니다.
 - 일반적으로 지속적으로 로봇을 움직이기 위해서는 초당 약 5 ~ 10회의 데이터 전송이 필요합니다.
 - 여기서는 10으로 정하겠습니다.

```
rate = rospy.Rate(10)
```


05 ROS Publisher / Subscriber

- ROS Publisher
 - 현재까지 작성한 내용은 다음과 같습니다.
 - 기본적으로 데이터를 보내기 위한 내용인 Node 등록, Publisher 생성, Publish할 속도 지정의 내용을 포함하고 있습니다.

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

rospy.init_node("turtle_drive_pub")
drive_pub = rospy.Publisher("cmd_vel", Twist, queue_size=10)
rate = rospy.Rate(10)
```

05 ROS Publisher / Subscriber

- ROS Publisher
 - 이제 실제 데이터를 보내는 부분을 작성하겠습니다.
 - 우선 Data를 ROS Master가 존재하는 동안 계속 보내야하므로, while을 이용해주어 지속적으로 동작하게 해주며, 보낼 메시지를 생성해줍니다.
 - 생성 후, 데이터를 보내고, 정해진 속도에 맞추어 대기하도록 합니다.

```
while not rospy.is_shutdown():  
    drive_msg = Twist((1.0, 0.0, 0.0), (0.0, 0.0, 1.0))  
    drive_pub.publish(drive_msg)  
    rospy.loginfo("Publish Data")  
    rate.sleep()
```

05 ROS Publisher / Subscriber

- ROS Publisher
 - 전체 코드는 다음과 같습니다.

```
#!/usr/bin/env python
import rospy
from geometry_msgs.msg import Twist, Vector3

rospy.init_node("turtle_drive_pub")
drive_pub = rospy.Publisher("cmd_vel", Twist, queue_size=10)
rate = rospy.Rate(10)

while not rospy.is_shutdown():
    drive_msg = Twist(Vector3(1.0, 0.0, 0.0), Vector3(0.0, 0.0, 1.0))
    drive_pub.publish(drive_msg)
    rospy.loginfo("Publish Data")
    rate.sleep()
```

05 ROS Publisher / Subscriber

- ROS Publisher
 - 이제 생성한 코드를 실행하도록 하겠습니다.
 - 우선 동작을 위해 turtlesim을 실행하겠습니다.
 - 첫 번째 터미널에서 `$ roscore`
 - 두 번째 터미널에서 `$ rosrn turtlesim turtlesim_node`

05 ROS Publisher / Subscriber

- ROS Publisher
 - 새로운 터미널을 열어 아래의 명령을 입력합니다.
 - `$ cd ~/catkin_ws`
 - `$ find ./src -name "*.py" -exec chmod +x {} \;`
 - `$ source devel/setup.bash`
 - `$ rosrn ros_tutorial turtle_drive_pub.py`

05 ROS Publisher / Subscriber

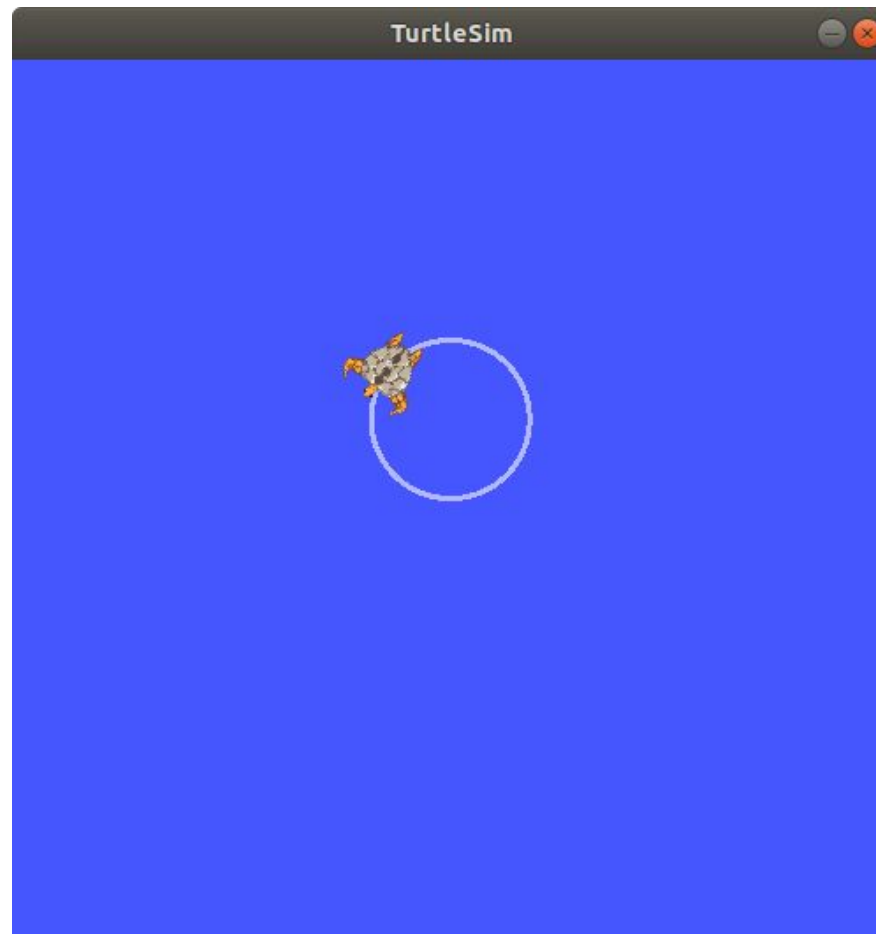
- ROS Publisher
 - 위의 명령을 입력하면 터미널에 Publish Data라는 내용이 출력되면서, 데이터를 보내게 되지만, 실제 Turtlesim의 거북이는 움직이지 않습니다.
 - 그 이유는 실제 Turtlesim을 움직이는 Topic은 /turtle1/cmd_vel 이지만, 우리가 생성한 Node는 /cmd_vel Topic에만 데이터를 보내고 있기 때문입니다.
 - 이 문제를 해결하기 위해서 간단히 코드를 수정하는 방법도 있지만, 다른 방법인 remap을 사용해보겠습니다.

05 ROS Publisher / Subscriber

- ROS Publisher
 - remap은 단순히, 특정 Topic의 이름을 다른 Topic으로 변경해주어 실행하는 역할을 도와줍니다.
 - 이전 페이지의 마지막 실행 부분을 다음과 같이 바꿔서 실행해보겠습니다.
 - `$ rosrn ros_tutorial turtle_drive_pub.py cmd_vel:=turtle1/cmd_vel`

05 ROS Publisher / Subscriber

- ROS Publisher
 - 그 결과 turtlesim이 다음과 같이 회전하는 것을 볼 수 있습니다.



05 ROS Publisher / Subscriber

- ROS Subscriber
 - 다음으로 Topic을 받는 Subscriber 코드를 작성해보겠습니다.
 - 이번에는 turtlesim에서 전달해주는 “turtle1/pose”를 받도록 하겠습니다.
 - 우선 해당 Topic의 정보를 확인하여, 메시지 타입을 확인해보겠습니다.
 - `$ rostopic info /turtle1/pose`

```
wego@wego-ubuntu:~/catkin_ws$ rostopic info /turtle1/pose
Type: turtlesim/Pose

Publishers:
 * /turtlesim (http://wego-ubuntu:45993/)

Subscribers: None

wego@wego-ubuntu:~/catkin_ws$
```

05 ROS Publisher / Subscriber

- ROS Subscriber
 - 또한 해당 메시지에서 확인할 수 있는 데이터를 확인해보겠습니다.
 - `$ rosmmsg show turtlesim/Pose`

```
wego@wego-ubuntu: ~/catkin_ws 79x21
wego@wego-ubuntu:~/catkin_ws$ rostopic info /turtle1/pose
Type: turtlesim/Pose

Publishers:
 * /turtlesim (http://wego-ubuntu:45993/)

Subscribers: None

wego@wego-ubuntu:~/catkin_ws$ rosmmsg show turtlesim/Pose
float32 x
float32 y
float32 theta
float32 linear_velocity
float32 angular_velocity

wego@wego-ubuntu:~/catkin_ws$
```

05 ROS Publisher / Subscriber

- ROS Subscriber
 - 해당 메시지에서는 x, y, theta, linear_velocity, angular_velocity를 담고 있습니다.

```
wego@wego-ubuntu: ~/catkin_ws 79x21
wego@wego-ubuntu:~/catkin_ws$ rostopic info /turtle1/pose
Type: turtlesim/Pose

Publishers:
 * /turtlesim (http://wego-ubuntu:45993/)

Subscribers: None

wego@wego-ubuntu:~/catkin_ws$ rosmmsg show turtlesim/Pose
float32 x
float32 y
float32 theta
float32 linear_velocity
float32 angular_velocity

wego@wego-ubuntu:~/catkin_ws$
```

05 ROS Publisher / Subscriber

- ROS Subscriber
 - 실제 받을 수 있는 데이터의 확인이 완료되었으니, 이제 데이터를 받아보겠습니다.
 - 코드는 turtle_pose_sub.py 의 이름으로 생성하겠습니다.
 - 우선 Publisher와 동일하게 최상단의 shebang 및 import를 해주겠습니다.
 - 메시지 타입에 맞게 import 해주시면 됩니다. 다음과 같습니다.

```
#!/usr/bin/env python
import rospy
from turtlesim.msg import Pose
```

05 ROS Publisher / Subscriber

- ROS Subscriber
 - 다음으로 해당 데이터를 받았을 때 수행하기 위한 callback함수를 하나 정의하겠습니다.
 - 여기서는 단순히, 데이터 출력만을 진행하겠습니다.

```
def callback(data):  
    rospy.loginfo("Subscribed Data = {}".format(data))
```

05 ROS Publisher / Subscriber

- ROS Subscriber
 - 이 후, 해당 Node를 ROS에 등록해줍니다. `rospy.init_node` 를 활용하겠습니다.
 - 다음으로 Subscriber 객체를 생성하겠습니다.
 - Subscriber 객체는 Topic 이름, 메시지 타입, Callback함수 순으로 입력을 받습니다.
 - Topic 이름은 “pose”, 메시지 타입은 Pose, Callback 함수는 생성한 callback으로 지정하겠습니다.

```
rospy.init_node("turtle_pose_sub")  
rospy.Subscriber("pose", Pose, callback)
```

05 ROS Publisher / Subscriber

- ROS Subscriber
 - 마지막으로 Subscriber는 지속적으로 Node를 유지해주며, Callback함수를 호출해주는 역할을 하는 함수인 `rospy.spin()`을 호출해주어야합니다.
 - 이를 포함한 전체 코드는 다음과 같습니다.

05 ROS Publisher / Subscriber

- ROS Subscriber

```
#!/usr/bin/env python
import rospy
from turtlesim.msg import Pose

def callback(data):
    rospy.loginfo("Subscribed Data = {}".format(data))

rospy.init_node("turtle_pose_sub")
rospy.Subscriber("pose", Pose, callback)
rospy.spin()
```


05 ROS Publisher / Subscriber

- ROS Subscriber
 - 작성한 코드를 실행해보겠습니다.
 - 이전과 동일하게 turtlesim, 이전에 생성한 Publisher도 돌아가고 있는 상태에서 추가적으로 아래 내용을 새로운 터미널에서 동작시켜줍니다.
 - `$ cd ~/catkin_ws`
 - `$ find ./src -name "*.py" -exec chmod +x {} \;`
 - `$ source devel/setup.bash`
 - `$ rosrn ros_tutorial turtle_pose_sub.py pose:=turtle1/pose`

05 ROS Publisher / Subscriber

- ROS Subscriber
 - Turtlesim이 움직이는 것에 따라, 결과가 달라지며 나오는 것을 볼 수 있습니다.

```
wego@wego-ubuntu: ~/catkin_ws 79x21
[INFO] [1655358466.714042]: Subscribed Data = x: 5.28027439117
y: 7.51106548309
theta: -2.89053630829
linear_velocity: 1.0
angular_velocity: 1.0
[INFO] [1655358466.729712]: Subscribed Data = x: 5.26484155655
y: 7.50684356689
theta: -2.87453627586
linear_velocity: 1.0
angular_velocity: 1.0
[INFO] [1655358466.746194]: Subscribed Data = x: 5.24947834015
y: 7.50237464905
theta: -2.85853624344
linear_velocity: 1.0
angular_velocity: 1.0
[INFO] [1655358466.762322]: Subscribed Data = x: 5.23418855667
y: 7.4976606369
theta: -2.84253621101
linear_velocity: 1.0
angular_velocity: 1.0
```

05 ROS Publisher / Subscriber

- ROS Publisher / Subscriber
 - 이와 같은 과정을 통해 Publisher와 Subscriber의 기본 코드를 확인해보았습니다.
 - 이 후에는 Subscriber와 Publisher를 하나의 Node에서 실행하거나, 다양한 알고리즘을 통해 동작하도록 수정해서 진행해보면 좋을 것 같습니다.



WeGo Robotics

Tel. 031 – 229 – 3553

Fax. 031 – 229 – 3554



제품 문의: go.sales@wego-robotics.com

기술 문의: go.support@wego-robotics.com