Tutorial 1 -
Ubuntu and ROS Introduction

MIE443- Aaron Hao Tan (Head Tutorial TA)

Developed by Daniel Dworakowski, Richard Hu,
Christopher Thompson and Aaron Hao Tan

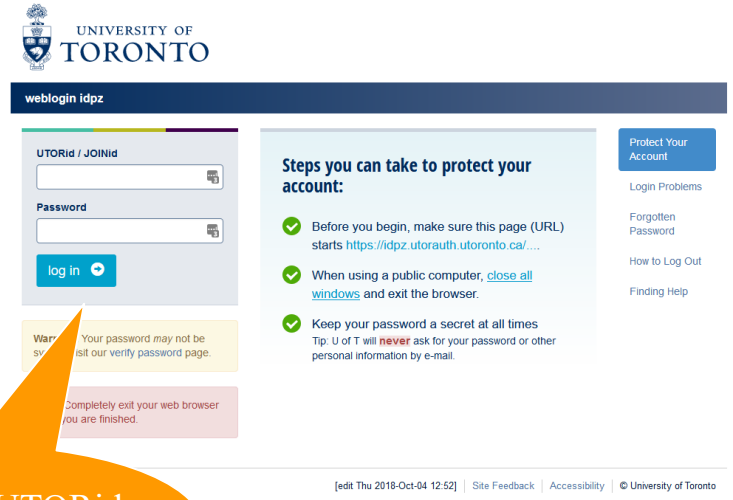
Important Note for all Contests!

- You are allowed to subscribe to any topic from the contest launch files (minimal.launch, gmapping_demo.launch etc.), however you are only allowed to publish to the topics specified in the *contest.cpp* files.
- Piazza Signup Link:

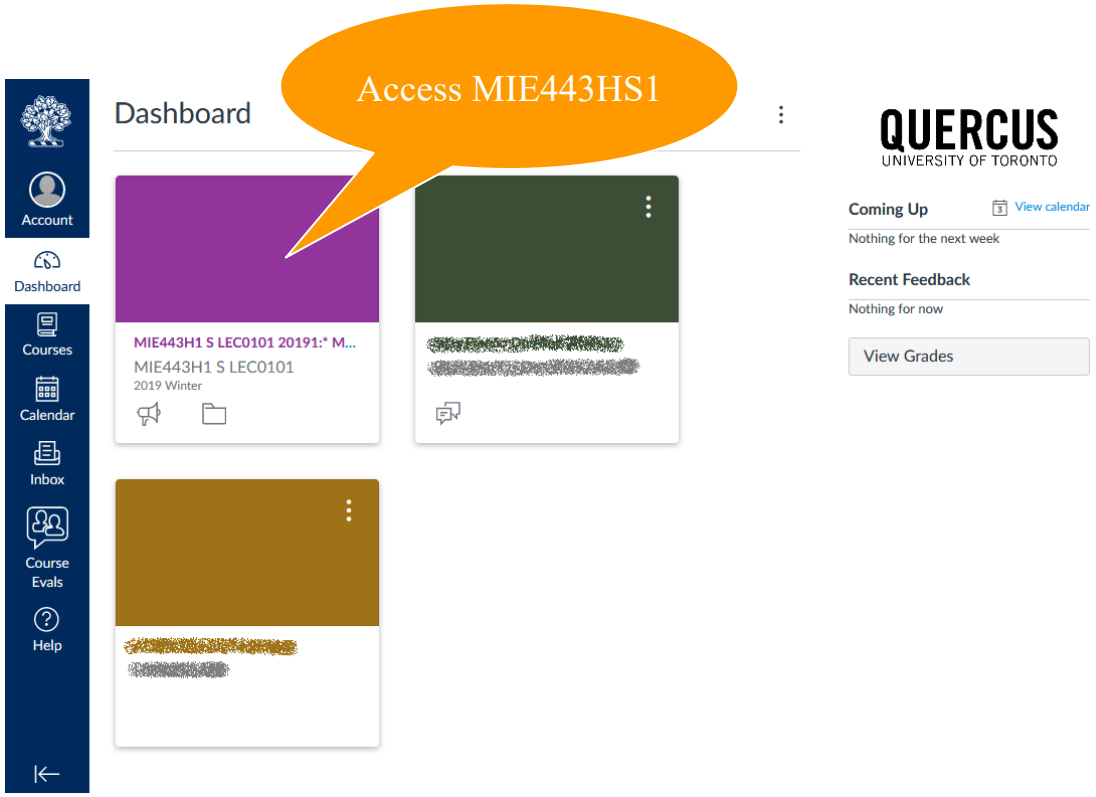
https://piazza.com/class/lbp5mhalunf49h

Downloading packages from Quercus

- Access Quercus (http://q.utoronto.ca/) and log-in to your account.
- On the Dashboard, access MIE443SH1 course area.

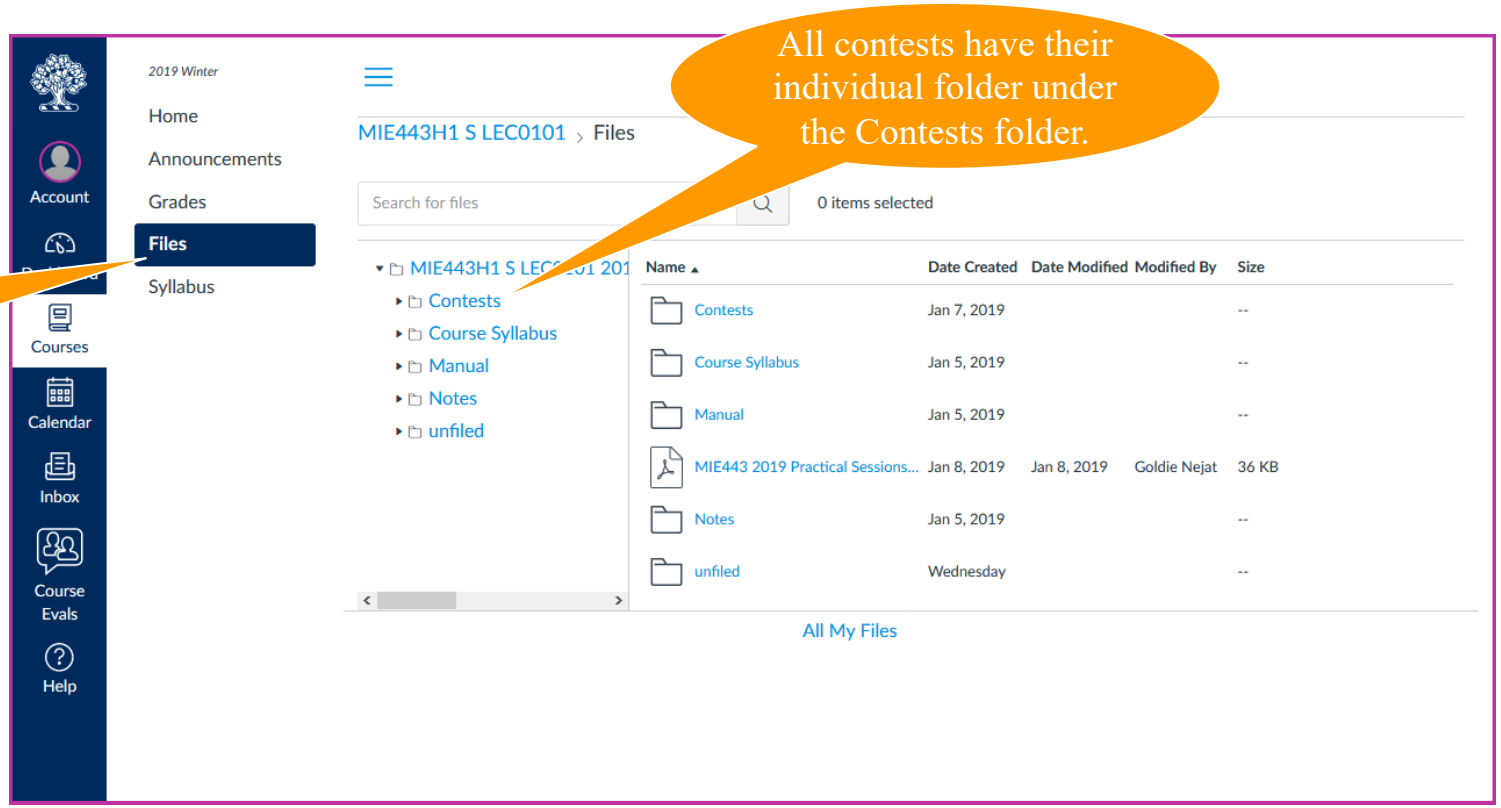


Enter your UTORid and password



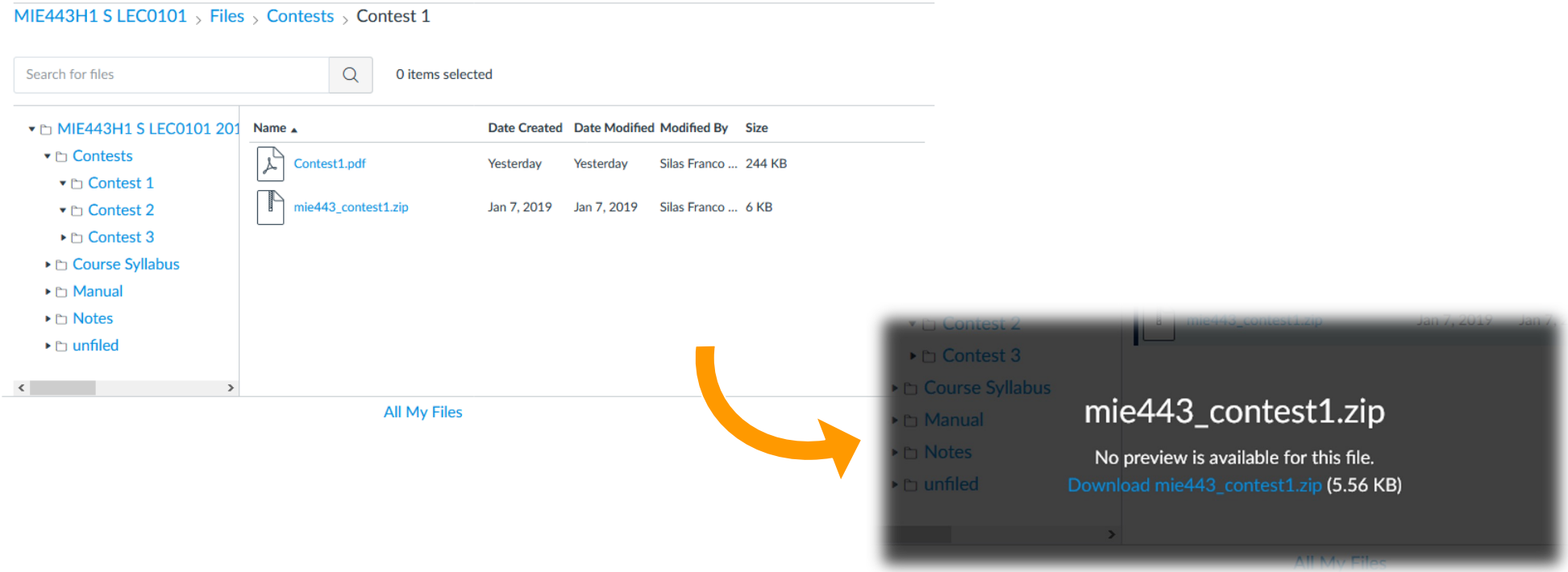
Accessing the Course Files

- On the course *Home* screen, click in *Files* to see all the documents and packages files uploaded to Quercus.
- Navigate to the *Contests* folder to access the source code and instructions for the contests.



Downloading Contest 1 Files

- On the *Files* section, select the *Contests* folder, then the *Contest 1* folder.
- Click on *mie443_contest1.zip* and then in *Download mie443_contest1.zip*.
- Also download turtlebot_script.sh from the Ubuntu setup folder in tutorial folder.



Ubuntu Setup

- The course requires Ubuntu 16.04 to be installed for ROS compatibility
- Multiple ways to install:

  1) Dual boot with windows

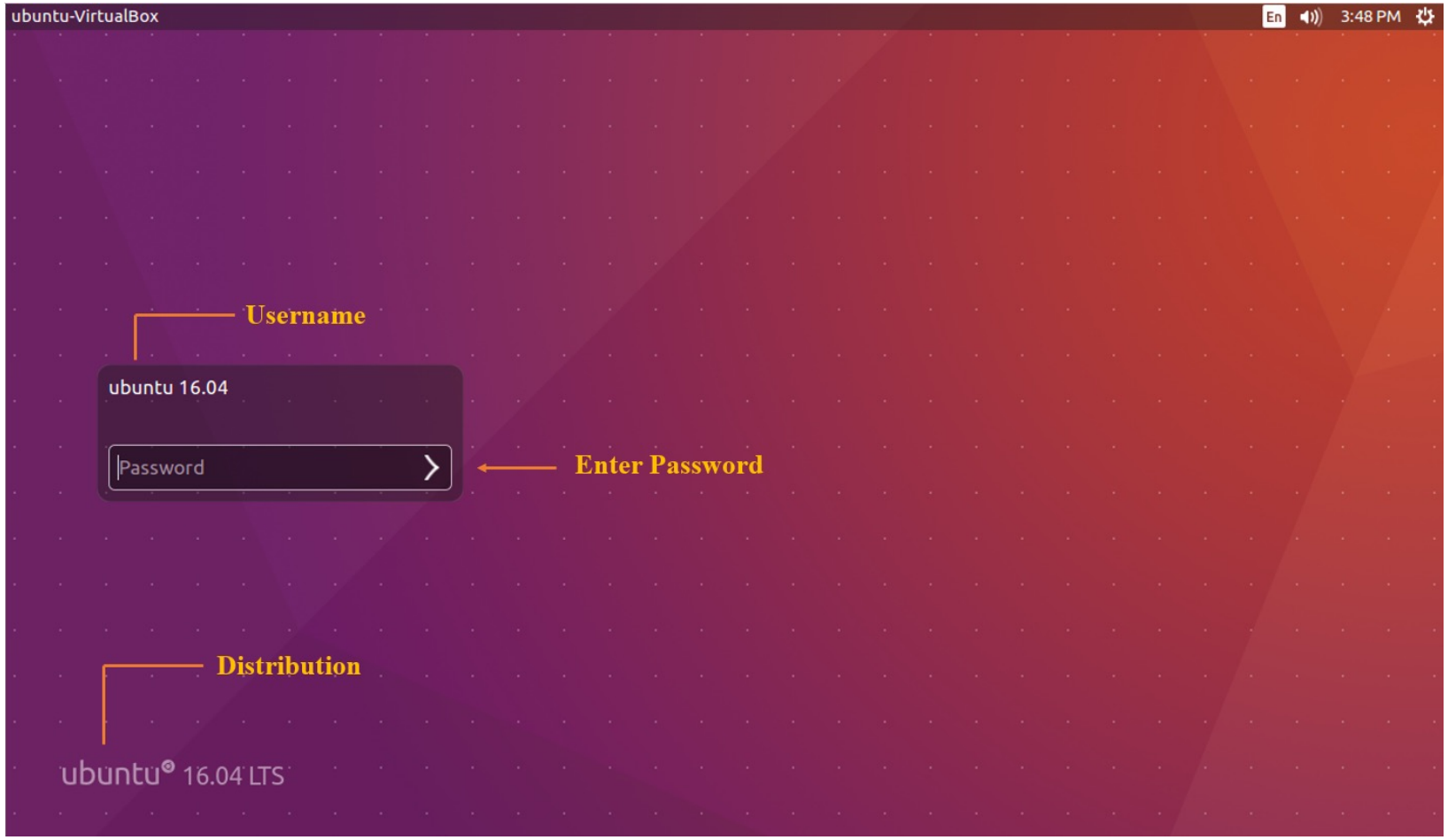
  2)Virtual box (see quercus for instructions “SettingUpUbuntuVirtualbox.pdf”)

  3) Docker (see quercus for instructions – for advanced users “SettingUpUbuntuDocker.pdf”)

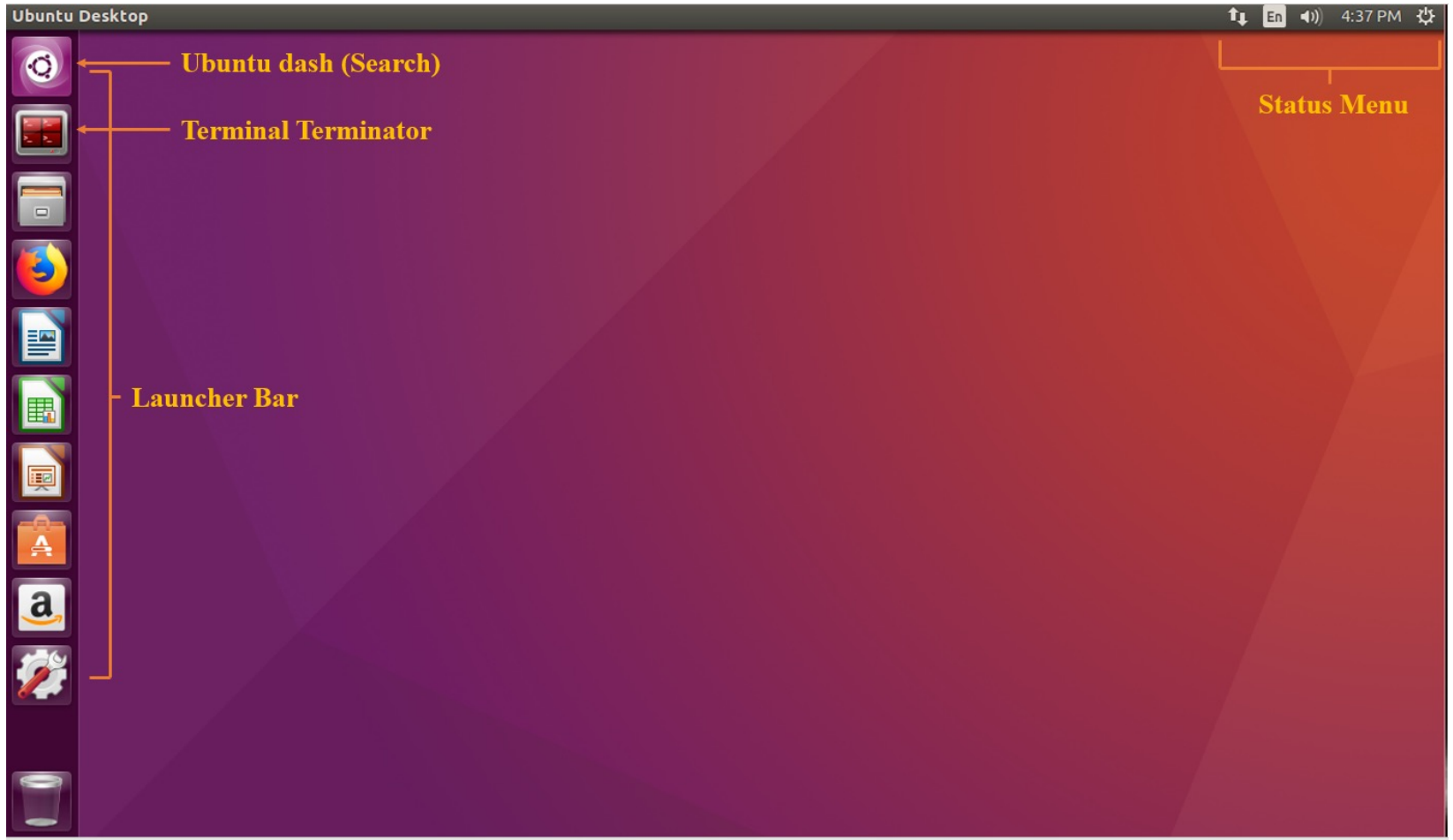
  – In general, users with Macs or Microsoft surfaces must install virtual machines
- You must then execute the turtlebot_script.sh script from quercus to install all basic software and settings
  – After downloading the file on Ubuntu open a terminal (alt+crtl+t) and enter:

```
bash ~/Downloads/turtlebot_script.sh #Press enter after copy-pasting this command and follow prompts
```

Login Screen

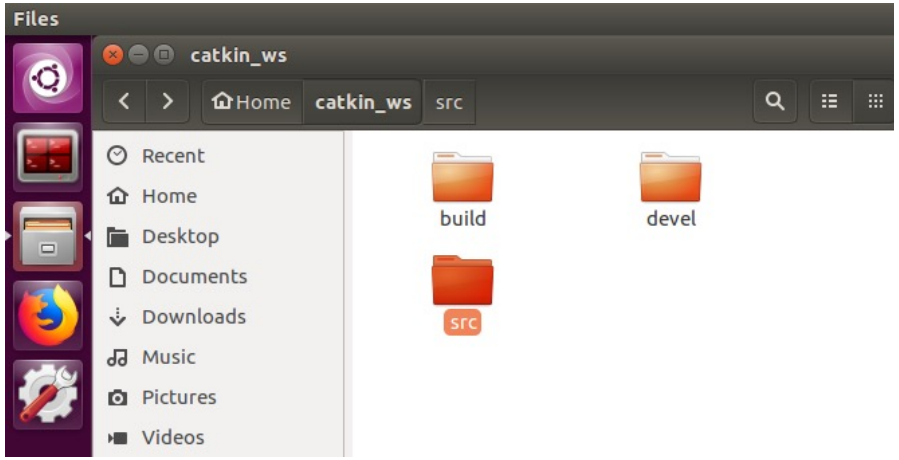


Desktop Screen and Launcher Bar

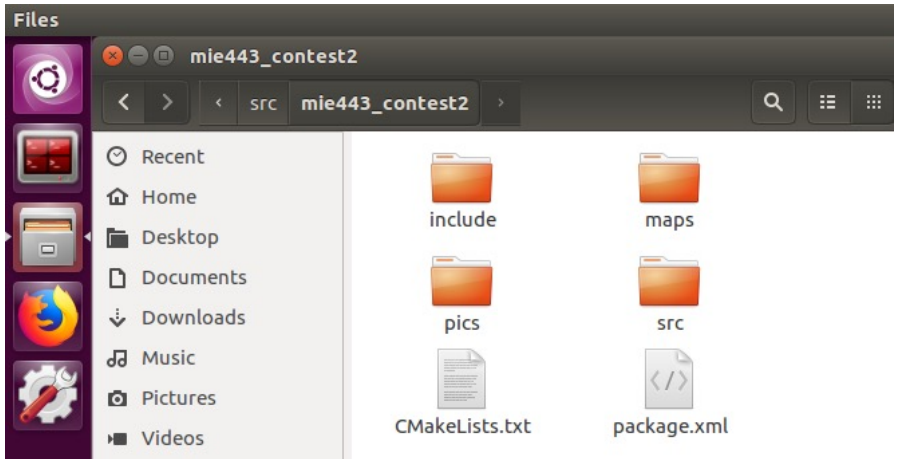


Workspace and Contest Packages

- Catkin Workspace
  - Directory containing all ROS files to be compiled
  - *src* folder contains all of the ROS packages
  - *build* and *devel* contain program dependencies and are automatically generated during compiling



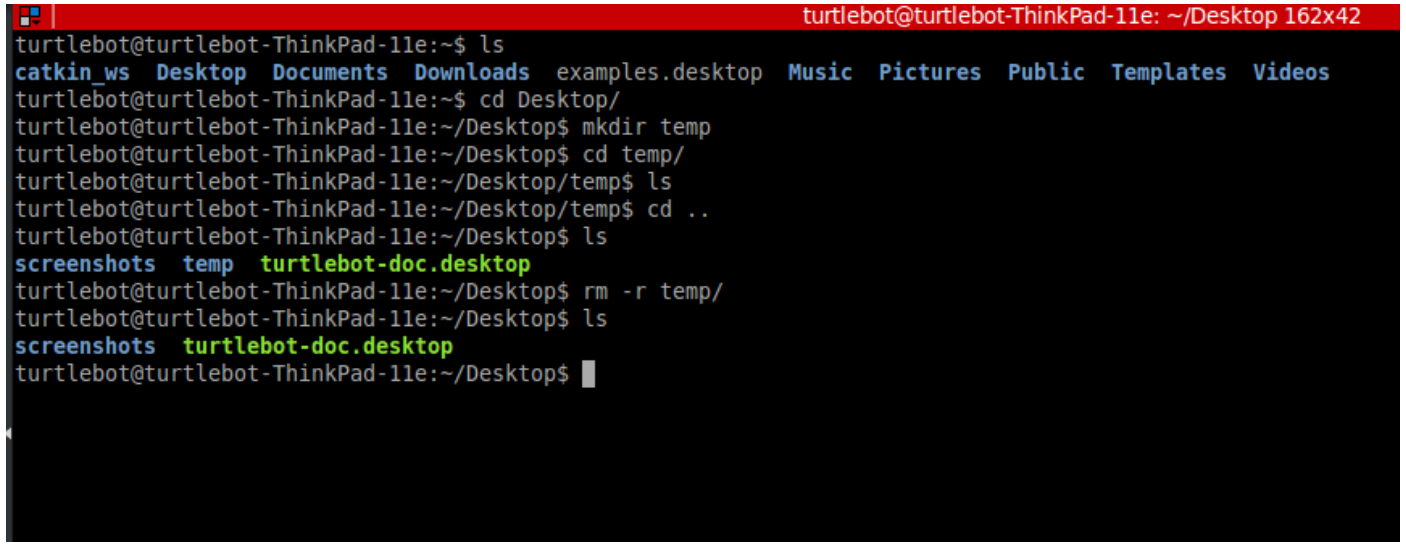
- Package Folders
  - Directories within the workspace that contain the resources to compile the package
  - Packages can contain many folders depending on complexity of the project



Terminal Interface

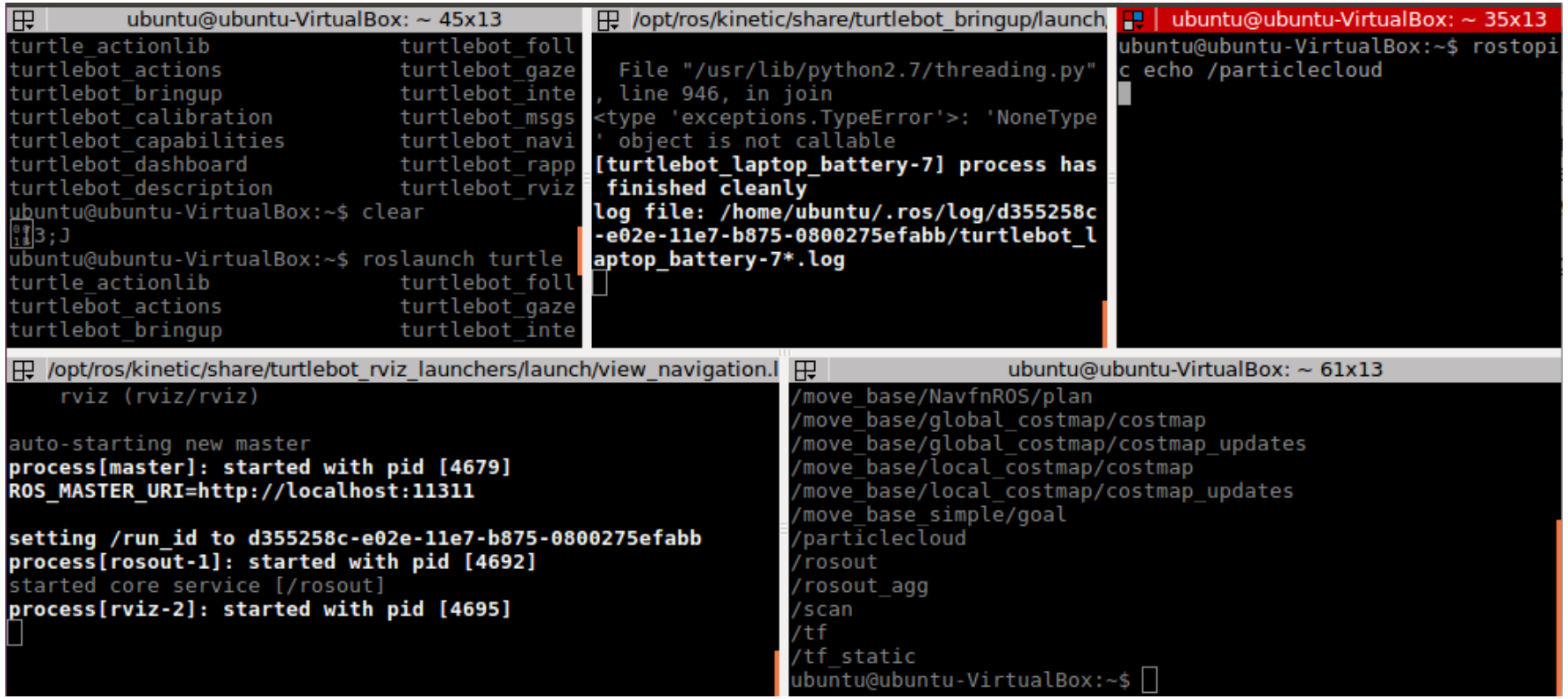
- Text-based/Non-Graphical interface
- Allows the user to navigate the computer system via text inputs and provides functionality that can be faster or is unavailable when using the graphical interface
- Command line input examples:

```
cd file/path # change directory
ls # list directory contents
mkdir name # create directory
```



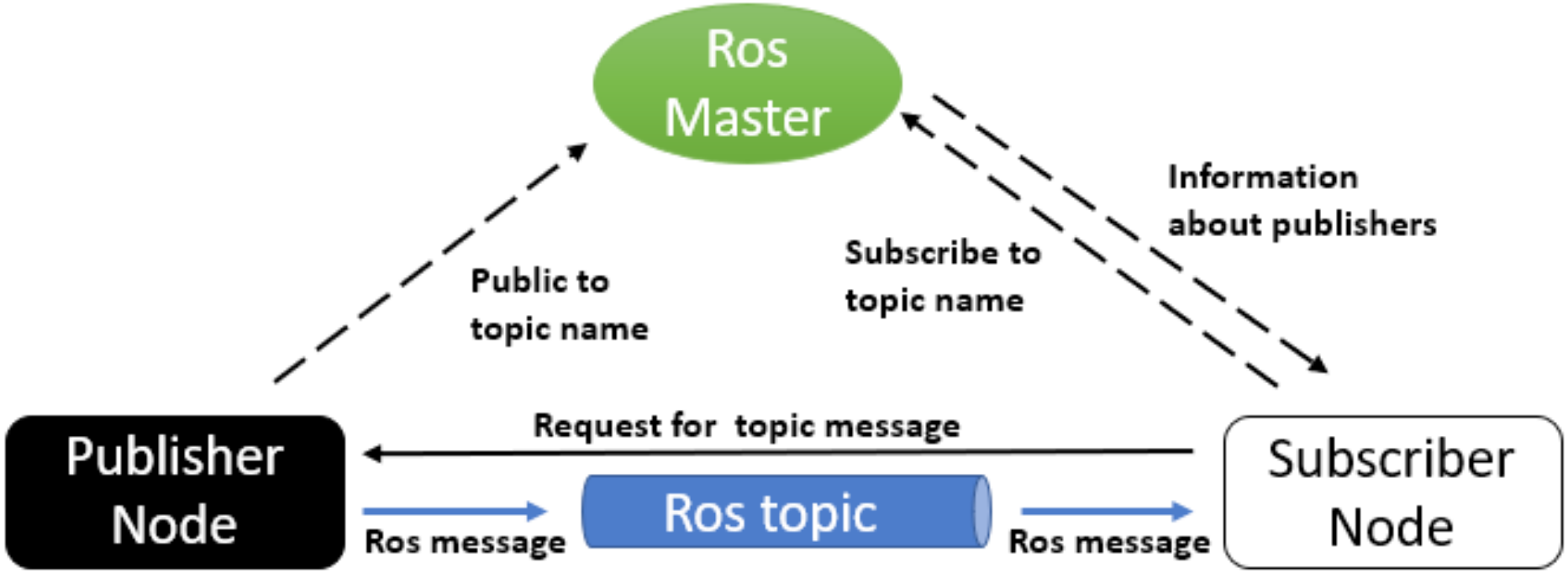
Terminal Emulator (Terminator)

- A wrapper program that extends the functionality of the basic terminal
  - Keyboard shortcut examples:
    - Ctrl-Shift-O – split screen up/down
    - Ctrl-Shift-E – split screen left/right
    - Ctrl-Shift-W – exit current window



ROS Overview

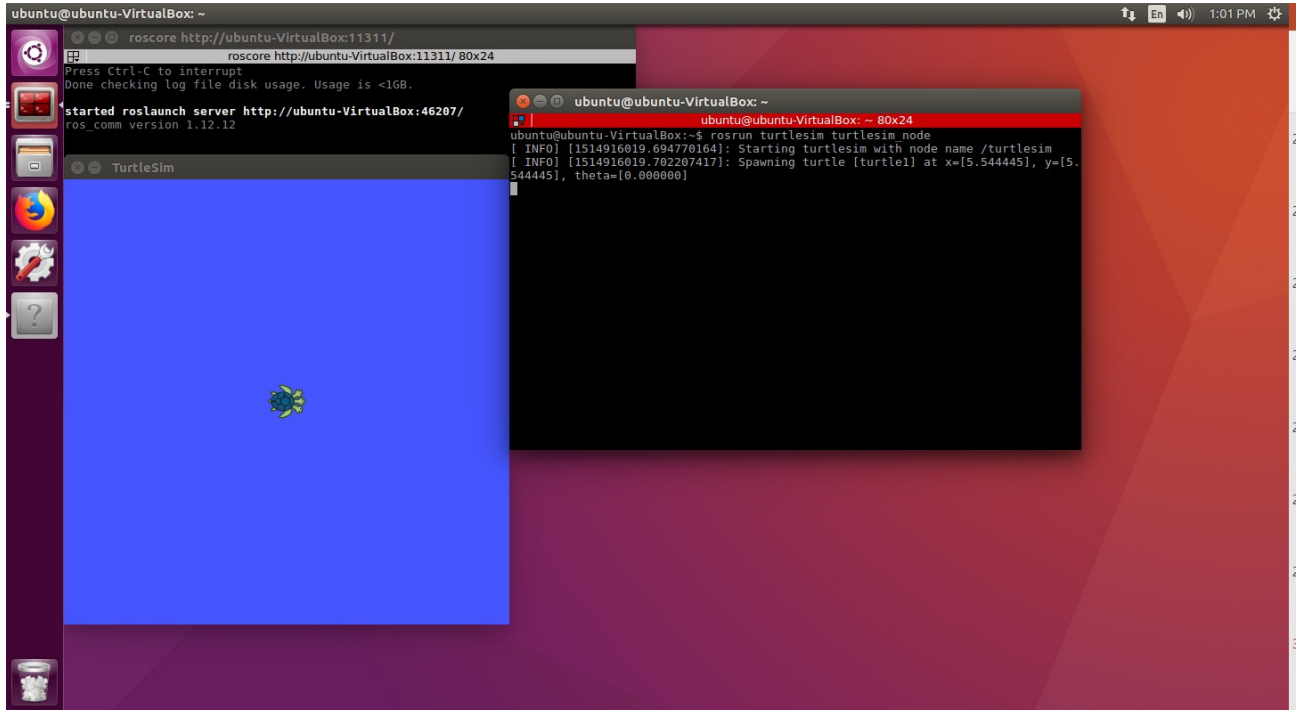
- ROS = Robot Operating System



ROS TurtleSim Example

- To run the TurtleSim example, the following commands have to be entered in their own respective terminals:

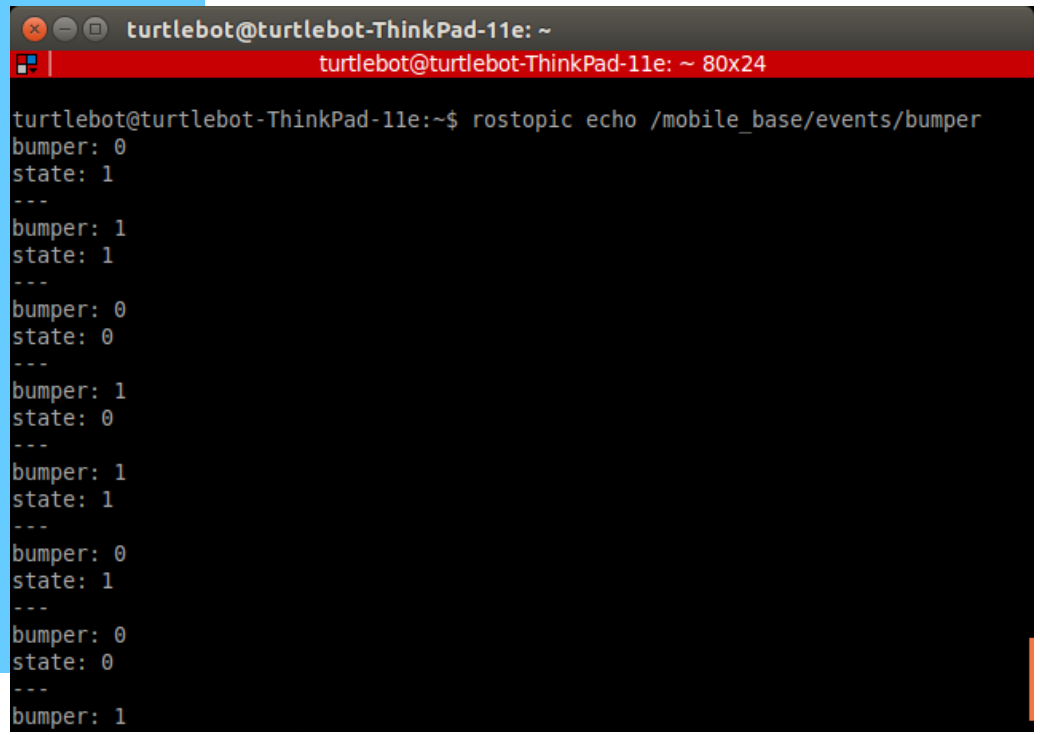
```
roscore #Initializes the ROS master
rosrun turtlesim turtlesim_node #Launches the turtlesim GUI
rosrun turtlesim turtle_teleop_key #Sends teleop commands to the turtlesim
```

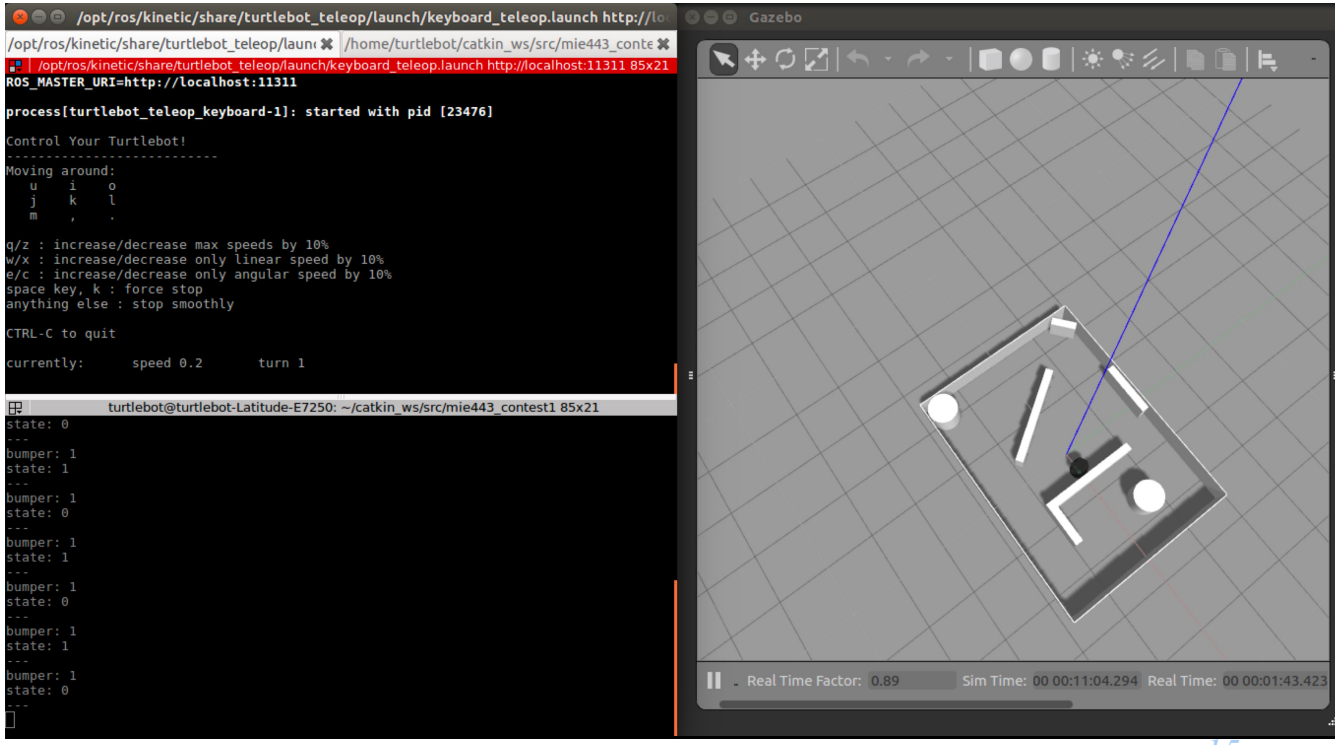


Demonstrating Topics

- The following commands illustrate how you can use the rostopic command to see the topic data passing between programs:

```
roslaunch mie443_contest1 turtlebot_world.launch world:=1 #Launch gazebo
roslaunch turtlebot_teleop keyboard_teleop.launch #Launch keyboard teleop
rostopic echo /mobile_base/events/bumper #listen to a published topic
```





Gmapping Demo

- The following commands illustrate how to use the simulated Turtlebot gmapping code to begin creating 2D maps of the environment:

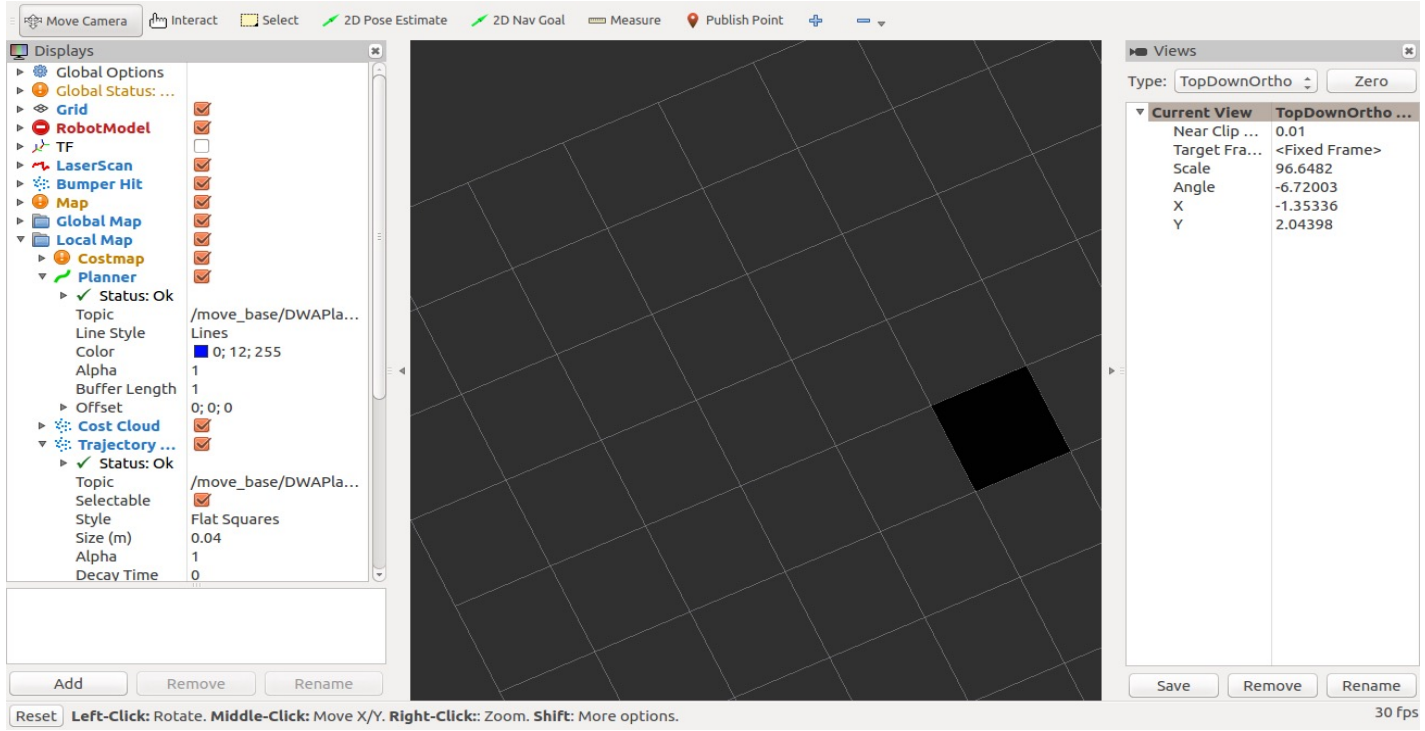
```
roslaunch mie443_contest1 turtlebot_world.launch world:=practice #Launch gazebo
roslaunch mie443_contest1 gmapping.launch #Begins the mapping software
roslaunch turtlebot_teleop keyboard_teleop.launch #begin Turtlebot base teleop software
```

```
/opt/ros/kinetic/share/turtlebot_gazebo/launch/gmapping_demo.launch http://localhost:11311 8
 * /slam_gmapping/minimumScore: 200
 * /slam_gmapping/odom_frame: odom
 * /slam_gmapping/ogain: 3.0
 * /slam_gmapping/particles: 80
 * /slam_gmapping/resampleThreshold: 0.5
 * /slam_gmapping/sigma: 0.05
 * /slam_gmapping/srr: 0.01
 * /slam_gmapping/srt: 0.02
 * /slam_gmapping/str: 0.01
 * /slam_gmapping/stt: 0.02
 * /slam_gmapping/temporalUpdate: -1.0
 * /slam_gmapping/xmax: 1.0
 * /slam_gmapping/xmin: -1.0
 * /slam_gmapping/ymax: 1.0
 * /slam_gmapping/ymin: -1.0

NODES
  /
    slam_gmapping (gmapping/slam_gmapping)

ROS_MASTER_URI=http://localhost:11311

process[slam_gmapping-1]: started with pid [7173]
```

Visualize Map

- To visualize various topics in ROS a program called Rviz is used. The following command is used to open Rviz and visualize the progress of the map being created by the simulated TurtleBot:

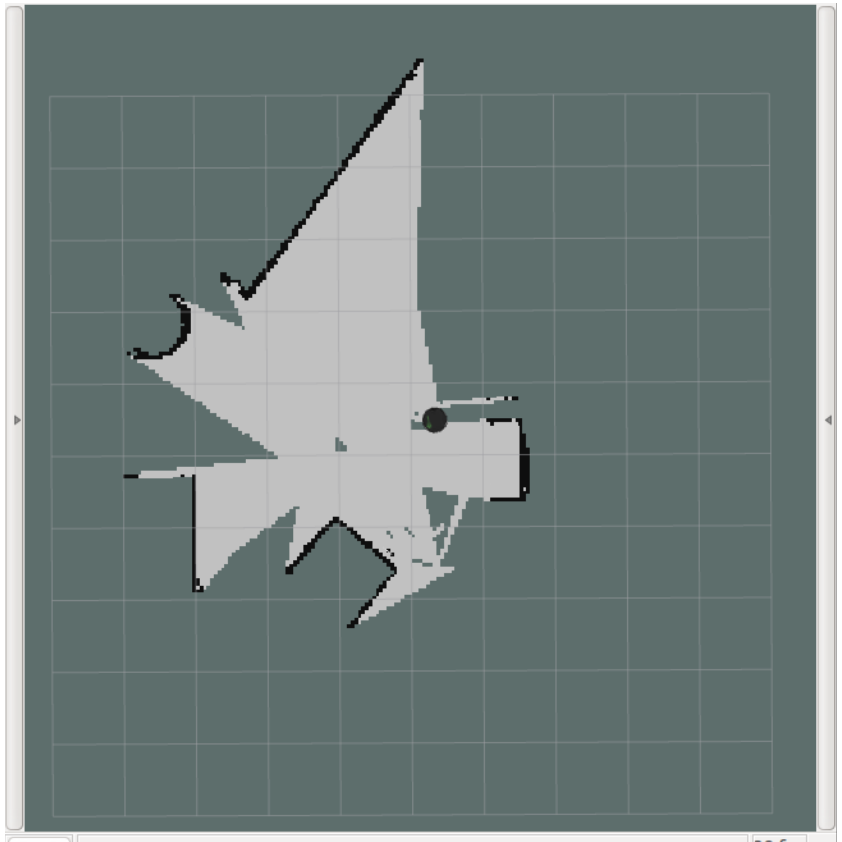
```
roslaunch turtlebot_rviz_launchers view_navigation.launch #Opens RVIZ that has turtlebot settings for visualizing a map
```



Saving a Map

- The following command is used to save a map being created with gmapping on the simulated Turtlebot for future use:

```
rosrun map_server map_saver -f your_map_name #Saves the gmapping map to the current directory with your own file name
```



Questions?

Talker & Listener Live Demo

1. Open a terminal: *ctrl* + *alt* + *t*
2. Make a catkin workspace folder and a subfolder called src

```
mkdir -p ~/talker_ws/src
```

3. Create a ros package called beginner_tutorials, that uses the libraries: std_msgs, rospy, and roscpp

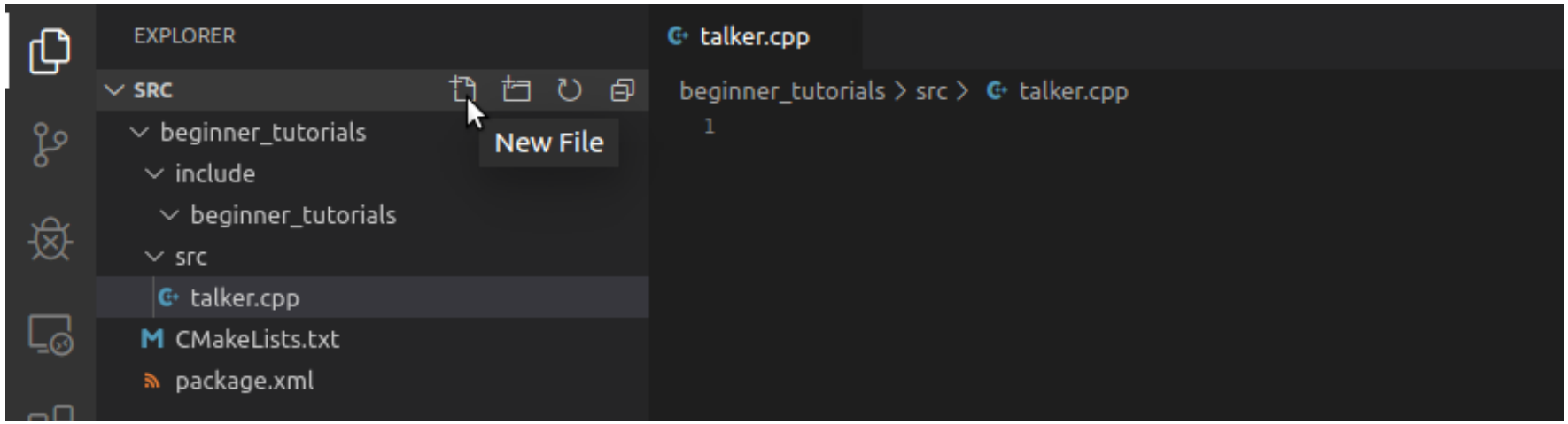
```
cd ~/talker_ws/src
catkin_create_pkg beginner_tutorials std_msgs rospy roscpp
```

4. Open VScode at this location, make sure to include the dot in the command

```
code .
```

Talker & Listener Live Demo

1. Create a *talker.cpp* file under *beginner_tutorials/src* folder, **alternatively**, this could also be done with the terminal command below:



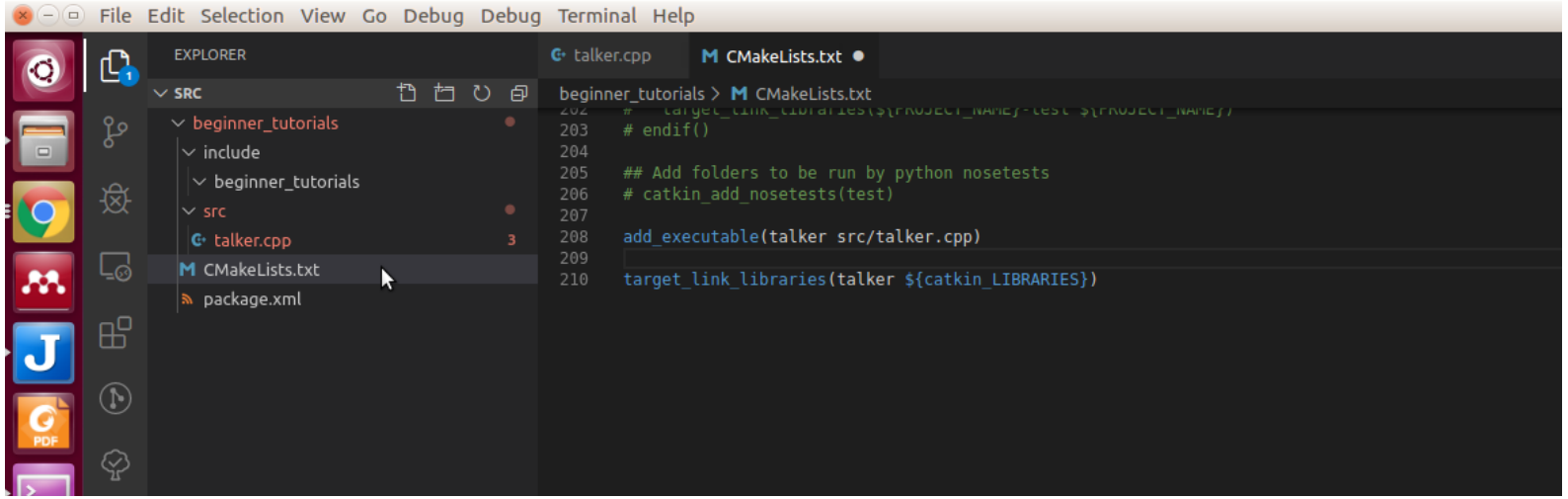

```
touch ~/talker_ws/src/beginner_tutorials/src/talker.cpp
```

2. Copy and paste the code from this link into the *talker.cpp* file:

https://raw.githubusercontent.com/ros/ros_tutorials/kinetic-devel/roscpp_tutorials/talker/talker.cpp

Talker & Listener Live Demo

1. Open the *CMakeLists.txt* of your *beginner_tutorials* package and append the compiling instructions to the bottom of the file:




```
add_executable(talker src/talker.cpp)
target_link_libraries(talker ${catkin_LIBRARIES})
```

2. Back in your terminal, build your workspace:

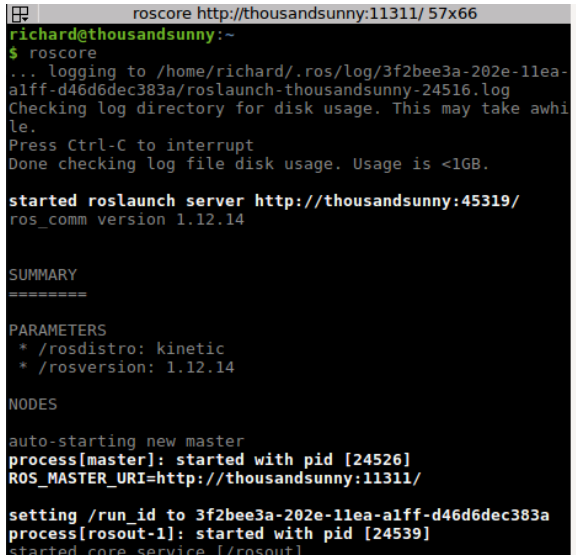
```
cd ~/talker_ws/
catkin_make
```

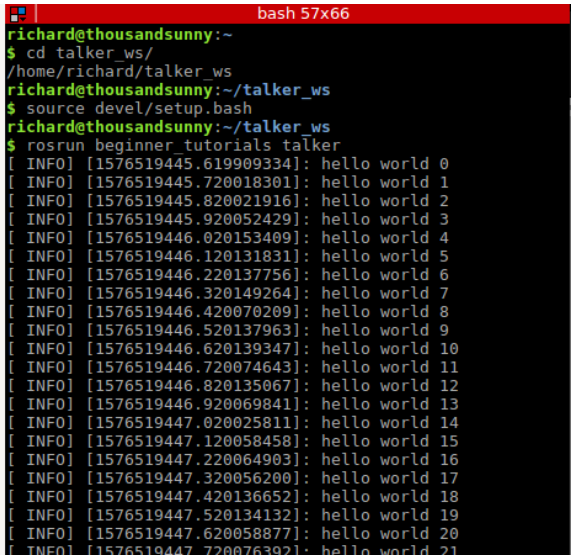
Talker & Listener Live Demo

1. Run *roscore* in a terminal.
2. In a **separate** terminal, follow the instruction to source the workspace you just built and run the talker node:

```
roscore #run this in terminal 1
```

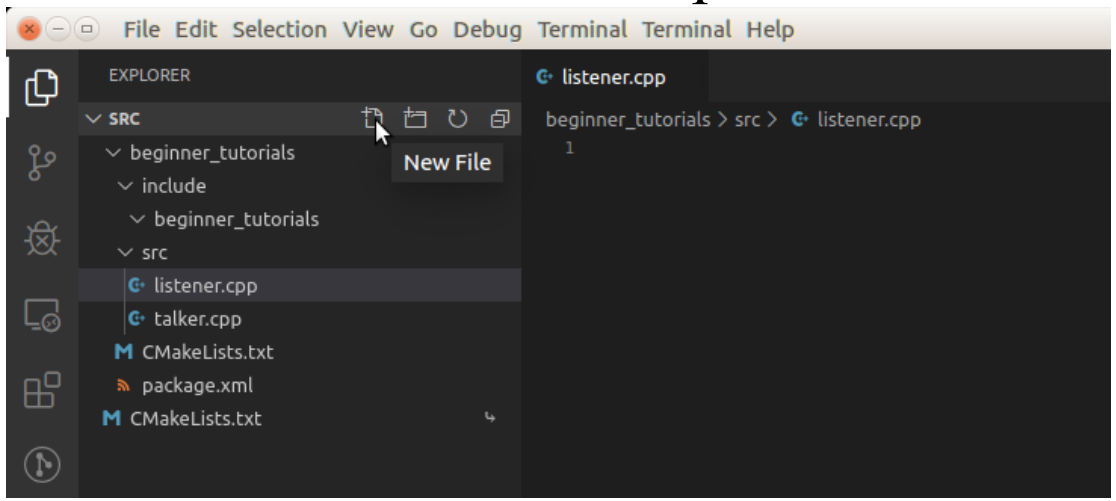
```
#run these in terminal 2
cd talker_ws
source devel/setup.bash
rosrun beginner_tutorials talker
```





Talker & Listener Live Demo

1. Create a *listener.cpp* under *beginner_tutorials/src* folder, similar procedure as the previous slides. Terminal command is provided below:

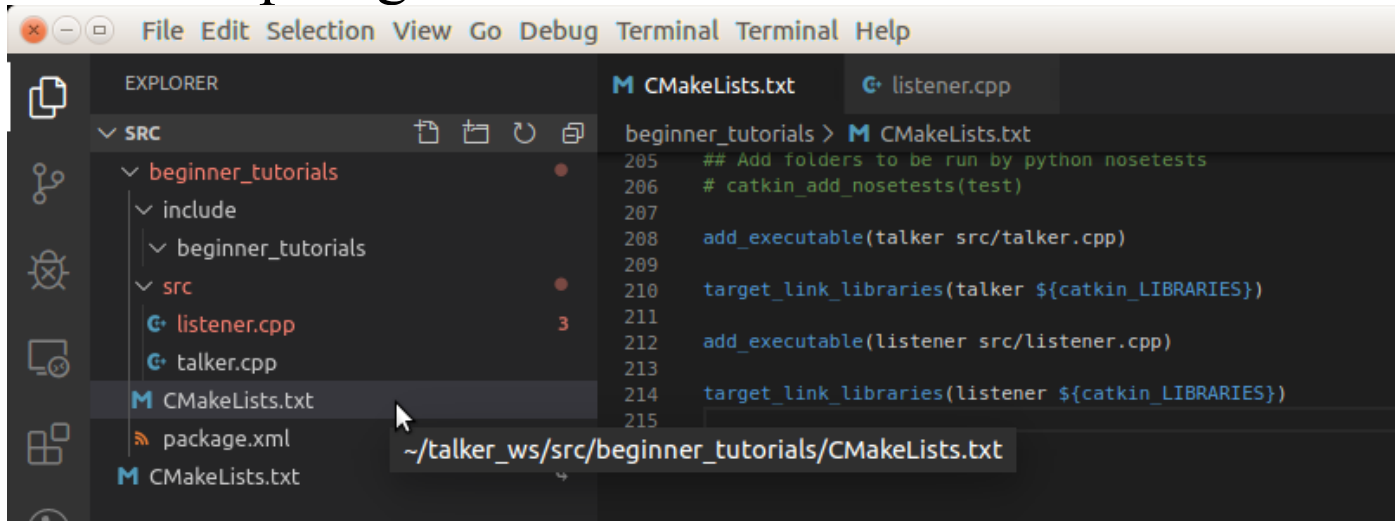



```
touch ~/talker_ws/src/beginner_tutorials src/listener.cpp
```

2. Copy and paste the code from this link into the *listener.cpp* file: https://raw.githubusercontent.com/ros/ros_tutorials/kinetic-devel/roscpp_tutorials/listener/listener.cpp

Talker & Listener Live Demo

1. Similar to before, open the *CMakeLists.txt* of your *beginner_tutorials* package and append the compiling instructions to the bottom of the file:




```
add_executable(listener src/listener .cpp)
target_link_libraries(listener ${catkin_LIBRARIES})
```

2. Back in your terminal, build your workspace:

```
cd ~/talker_ws/
catkin_make
```

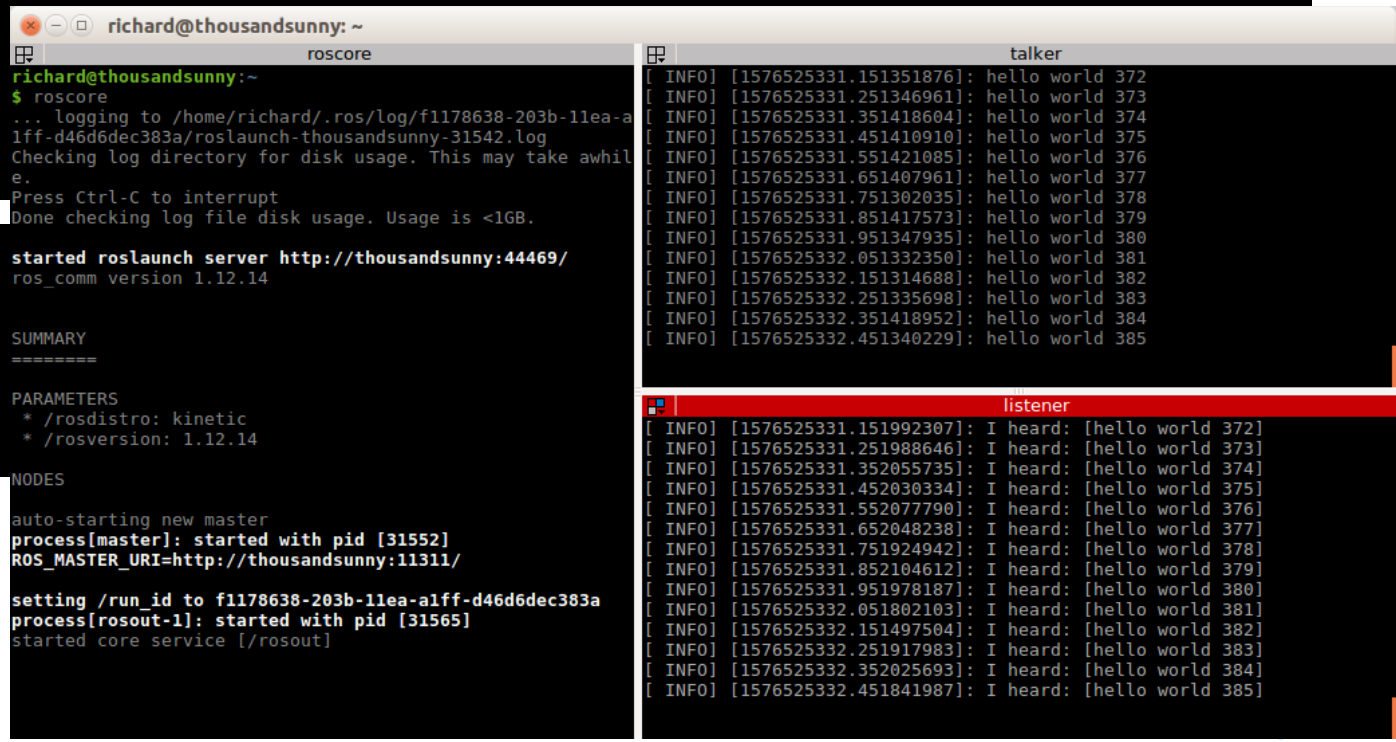
Talker & Listener Live Demo

1. Run *roscore* in a terminal.
2. In a **separate** terminal, follow the instruction to source the workspace you just built and run the talker node:

```
roscore #run this in terminal 1
```

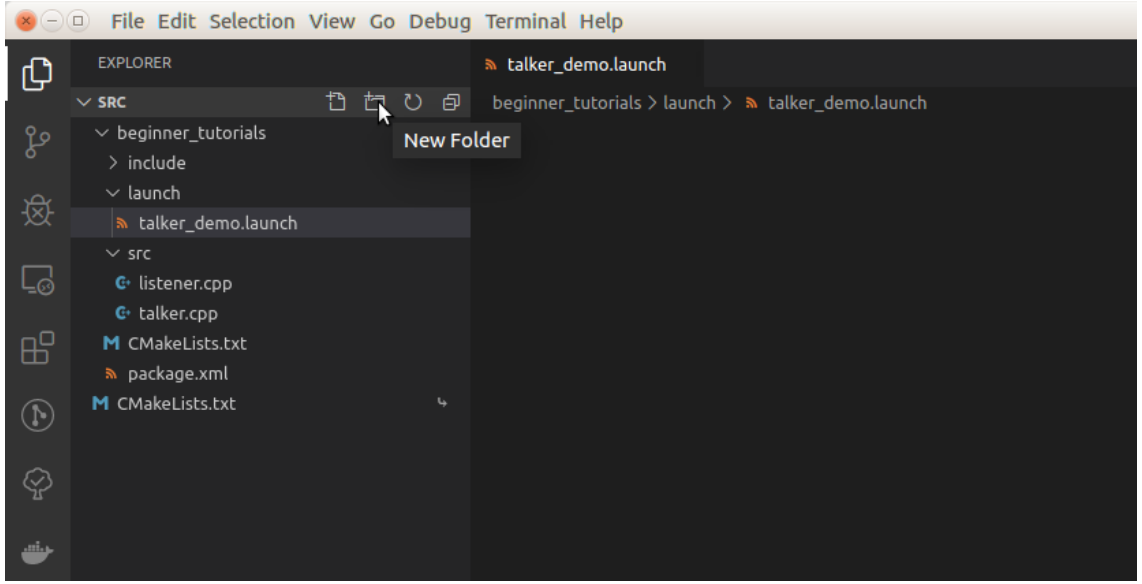
```
#run these in terminal 2
cd talker_ws
source devel/setup.bash
rosrun beginner_tutorials talker
```

```
#run these in terminal 3
cd talker_ws
source devel/setup.bash
rosrun beginner_tutorials listener
```



Talker & Listener Live Demo

1. Create a folder called *launch* under *beginner_tutorials,* and a file called *talker_demo.launch* in the *launch* folder:



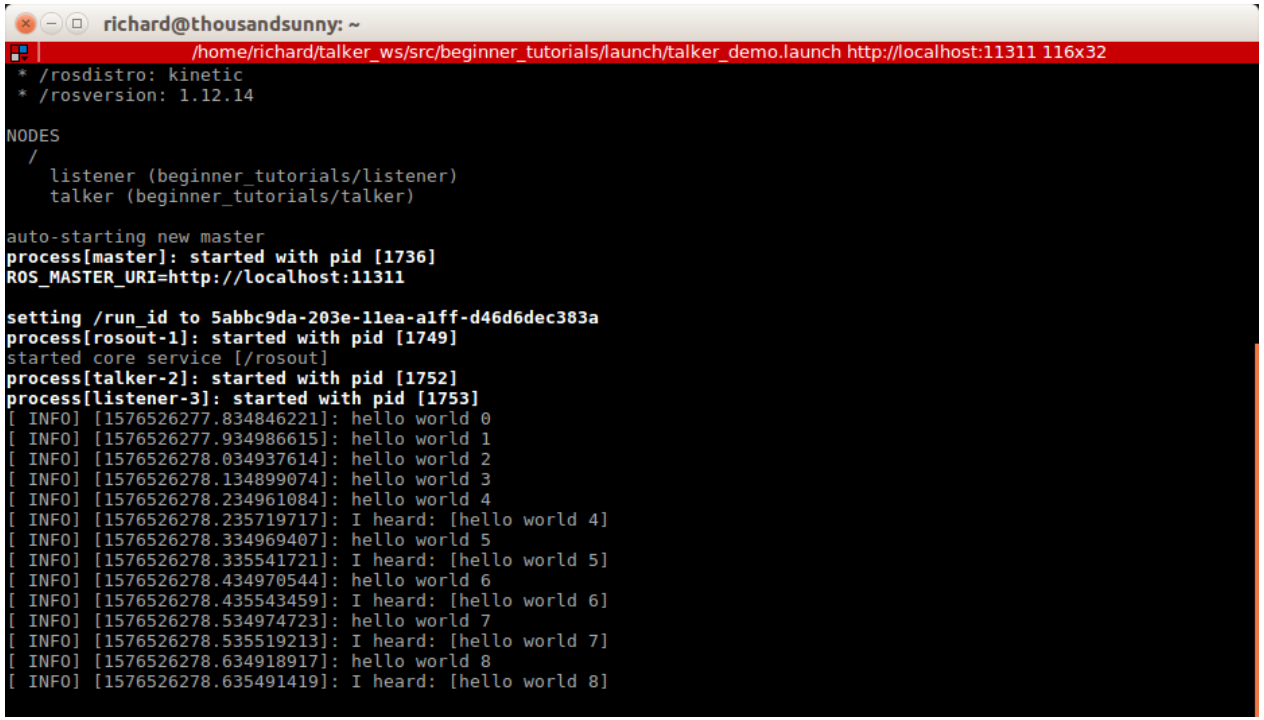
2. Type in the following lines into the *talker_demo.launch* file:

```
<launch>
  <node name="talker" pkg="beginner_tutorials" type="talker" output="screen"/>
  <node name="listener" pkg="beginner_tutorials" type="listener" output="screen"/>
</launch>
```

Talker & Listener Live Demo

1. You can launch several nodes and a roscore at the same time using a launch file. Follow the instruction to run the launch file, note that you do not need to rebuild your workspace nor to run a *roscore* for these steps:

```
cd talker_ws
source devel/setup.bash
roslaunch beginner_tutorials talker_demo.launch
```



Questions?

Visual Studio Code C++ Linter

- C++ Linter will show you the errors in your code.

```
ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);

ros::Rate loop_rate(10);

int count = 0;
while (ros::ok())
{

  std_msgs::String msg;

  std::stringstream ss;
  ss << "hello world " << count;
  msg.data = ss.str();

  ROS_INFO("%s", msg.data.c_str());
```

PROBLEMS SEARCH OUTPUT DEBUG CONSOLE Filter. E.g.: text, **/*.ts, !**/node...

No problems have been detected in the workspace so far.

```
ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);

ros::Rate loop_rate(10);

int count = 0;
while (ros::ok())
{
  making an error here

  std_msgs::String msg;

  std::stringstream ss;
  ss << "hello world " << count;
  msg.data = ss.str();
```

PROBLEMS 3 SEARCH OUTPUT DEBUG CONSOLE Filter. E.g.: text, **/*.ts, !**/no...

talker.cpp beginner_tutorials/src 3

- identifier "making" is undefined [23, 5]
- expected a ';' [23, 15]
- identifier "msg" is undefined [29, 5]

Visual Studio Code C++ Linter

- Setting up a Linter

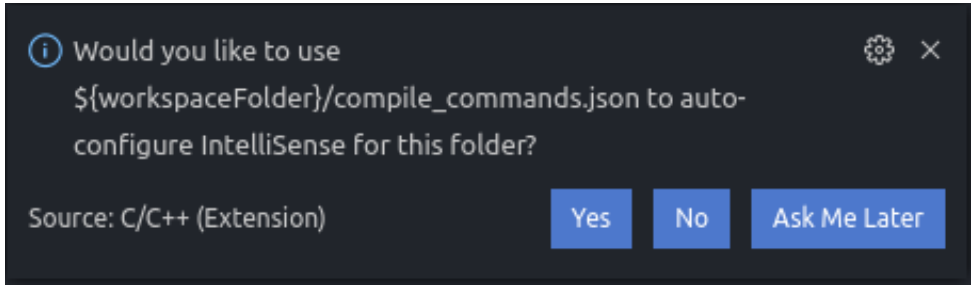
1. run the following command to generate compile_commands.json, as well as moving it under your catkin_ws/src folder:

```
catkin_make -DCMAKE_EXPORT_COMPILE_COMMANDS=ON && mv build/compile_commands.json src/
```

2. open your workspace in vscode, the easiest way to do it is in terminal:

```
code ~/catkin_ws/src/
```

3. Click yes when this window pops up:



Notes:

1. You only need to do step 1 once, unless you add new libraries or new packages to your code.
2. First time using the linter will take a minute to parse your files, it will be faster after.
3. The linter detects changes when you save your file.