**Mokpo National University**

**Debul Campus**

**Debul, Mokpo**

# Robot Operating system: A Basic Lab Operating Book

**By : Shree Krishan Acharaya**

**Corresponding: Prof. Sol Ha**

**Chapter 1:**

# Introduction to Robot operating system:

The Robot Operating System (ROS) is a flexible framework for writing robot software. It is the collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. Creating truly robust, general-purpose robot software is hard because human are vary wildly between instance of tasks and environments but it seem trivial to the robot. To deal with this problems, ROS was built to encourage collaborative robotics software development. Many groups expertise are navigate and contribute to each other in world –class systems.

## 1.1 ) ROS contribution:

ROS is an open source project, and the code with it is the result of the combined efforts of an international community.

## 1.2 ROS History

ROS is a large project with many ancestors and contributors. The need for an open-ended collaboration framework was felt by many people in the robotics research community, and many projects have been created towards this goal. In mid-of 2000s, various efforts at STanford University became crucial for dynamic software systems intended for robotics use. Such created flexible prototypes involved integrative and embodied AI such as the STanford AI Robot (STAIR) and the Personal Robots (PR) program. A visionary robotics incubator Willow George, in 2007, provides significant resources to extend those concepts with tested implementations. Then effort is boosted by countless expertise with core ROS idea and its fundamental packages. Nowadays any group can start their own ROS code repository on their own servers, and they maintain full ownership and control it. The ROS ecosystem now consists of a large number of users in worldwide, working in domains ranging from tabletop hobby projects to large industrial automation systems.

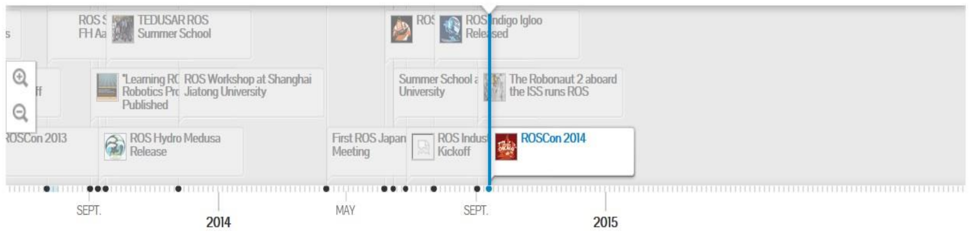

ROS S FH Aa
TEDUSAR ROS Summer School
ROS
ROS Indigo Igloo Released
"Learning RC Robotics Prc Published
ROS Workshop at Shanghai Jiatong University
Summer School a University
The Robonaut 2 aboard the ISS runs ROS
ROSCon 2013
ROS Hydro Medusa Release
First ROS Japan Meeting
ROS Indus Kickoff
ROSCon 2014
SEPT.
2014
MAY
SEPT.
2015

*Figure 1: The third ROS developers' conference was held in Chicago, Illinois*

## 1.3 ROS Core Components:

Some core part of ROS are identified and talk about their functionality , technical specifications, and quality in order to give a better idea of what ROS can contribute to your project.

- Communication Infrastructure:
- Message passing

- Recording and Playback of message
- Remote Procedure calls
- Distributed Parameter System
- Root –specific Feature
- Standard Robot Message
- Robot Geometry Libery
- Robot Description Language.
- Preemptalble Remothe calls
- Diagonstics
- Pose Estimation , Localization, and Navigation
- Tools
- Command –Line Tools
- Rviz
- Rqt

## 1.4 ROS support

There are several mechanisms in place to provide support to the ROS community, each with its own purpose: the wiki, ROS Answers, issue trackers, and the ros-users@ mailing list. It is important to pick the right resource to reduce response time, avoid message duplication, and promote the discussion of new ideas.

- **Wiki:**
  When something goes wrong, the wiki is your first stop. In addition to the official documentation for ROS packages, the wiki contains two key resources you should consult: The **troubleshooting guide** and the **FAQ**. Solutions of too many problems are covered in these two pages.
- **ROS Answers :**
  If the wiki doesn't address your problem, ROS answer is next. It is very likely that someone else has faced the same problem before, and that it's covered among the more than 10,000 questions at ROS answer. Start by searching for questions similar to yours; if your question isn't already asked, post a new one. Be sure to check the guidelines on how to prepare your questions before question.
- **Issue Trackers:**
  When you've identified a bug (e.g. as a result of a discussion ROS answers), of when you want to request a new feature, head to the issue trackers. When reporting a bug, be sure to provide a detailed description of the problem, the environment in which it occurs, any detail that may help developers to reproduce the issue , and if possible , a debug backtrack.
- **Ros-users@**
  To stay up-to-date on the latest developments within the ROS community, you'll want to subscribe to the ***ros-users@mailing list***. It has thousands of members, which is paklcae of announcements, news and discussions of general interest. The **ros-user@list** is not the right place to ask troubleshooting questions of reports bugs.

## 1.5 ROS contribute:

ROS is a commonly-driven project that represents the combined efforts of many, many contributions.

- **Join the family:**
  Since we are a newcomer to ROS community, one of the first ways to contribute is to join the ROS Answers Q&A forum and participate, both by asking your own questions and answering other's question when we can.
- **Open Issues(and Provide Patches)**
  If we are using a piece of open source ROS software and run into a problem, It is very helpful to the community if we open an issue against the package to make sure that the developers are aware of the problem. If we have ability to track down the problem and create a patch that fixes it.

- **Develop and Share New capabilities**
  It requires a specific user, whether it was for an experiment, a demonstration, or a product. Due to releasing such products to the community helps to the new comers. A useful capability can quickly be taken up by thousands of ROS users around the world.
- **Maintain Packages**
  Package maintainer's plays important role who release packages into specific ROS distributions and verify integration. They did commitment to do new development packages. They pick up a package and take responsibility for keeping it available to the greater ROS community.

**Chapter 2**

# Installation of ROS software with RaspberryPI

The raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard Keyboard and Mouse. It is a capable device where we can learn programming like Python and Scratch, browsing internet though Wi-Fi or Ethernet, playing video and games and making word-processing and spread-sheets.

**Some basic command for dos operating system:**

I) Make directory
Command: ***mkdir directory_name***

II) Go to directory
Command: ***cd directory_name***

III) Lookup directory
Command: ***ls***

IV) Remove directory
Command : ***rm –r –f directory_name***

V) Create file
Command: ***nano file_name***

VI) Remove file
Command: ***rm file_name***
Copy or move directory
Command: ***cp or mv source_filename destination_filename***

**VII)**
**VIII)**
**IX)**

**2.1) Installing ROS on the Raspberry Pi:**

- Installing **ROS Fuerte**:
  Fdgfg ghg
  http://wiki.ros.org/ROSberryPi/Installing%20ROS%20Fuerte%20on%20RaspberryPi
- **Installing ROS Groovy:**
  **Dfji f** kdf f
  http://wiki.ros.org/groovy/Installation/Raspbian
- **Installing ROS Hydro:**
  **Fdfdfdrf**
  http://wiki.ros.org/ROSberryPi/Setting%20up%20Hydro%20on%20RaspberryPi
- **Installing ROS Indigo:**
  **Fgfg**
  http://wiki.ros.org/ROSberryPi/Installing%20ROS%20Indigo%20on%20Raspberry%20Pi

## 2.2) Installing steps of ROS Indigo in RaspberryPI:

**a) Prerequisites:**

These instructions assume that Raspbian is being used as the OS on the Raspberry Pi. We download image of Raspbian from [http://www.raspberrypi.org/downloads/](http://www.raspberrypi.org/downloads/). The latest version of Raspberry pi OS is NOOBS and Raspbian Jessie. NOOBS is easy to install and make for beginner. We can simply download and put in memory card. After hardware setting up to the Raspberry we found installation steps as similar to the windows setup. However, Raspbian Jessie is advanced OS of raspberry pi, it has image file so we have to make bootable memory card and install it.

## Set up ROS Repositories

*Raspbian Wheezy:* % for NOOBS is Wheezy is replace by NOOBS

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu wheezy main" >
/etc/apt/sources.list.d/ros-latest.list'
$ wget https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -O - |
sudo apt-key add -
```

*Raspbian Jessie:*

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu jessie main" >
/etc/apt/sources.list.d/ros-latest.list'
$ wget https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -O - |
sudo apt-key add -
```

Now, make sure your Debian package index is up-to-date:

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

## Install Bootstrap Dependencies

*Raspbian Wheezy:*

```
$ sudo apt-get install python-pip python-setuptools python-yaml python-
argparse python-distribute python-docutils python-dateutil python-six
$ sudo pip install rosdep rosinstall_generator wstool rosinstall
```

*Raspbian Jessie:*

```
$ sudo apt-get install python-pip python-setuptools python-yaml python-
distribute python-docutils python-dateutil python-six
$ sudo pip install rosdep rosinstall_generator wstool rosinstall
```

### Initializing rosdep

```
$ sudo rosdep init
$ rosdep update
```

## b) Installation

Now, we are going to download and build ROS Indigo

**Creating a catkin Workspace**

Catkin Workspace used for build the core packages. We can create one by using this command:

```
$ mkdir ~/ros_catkin_ws
$ cd ~/ros_catkin_ws
```

We use **wstool** for fetch the core packages which is useful to build them. We have many types of **wstool** as particular that we want to install.

**ROS-Comm: (recommended)** ROS package, build, and communication libraries. No GUI tools.

- `$ rosinstall_generator ros_comm --rosdistro indigo --deps --wet-only --exclude roslisp --tar > indigo-ros_comm-wet.rosinstall`
- `$ wstool init src indigo-ros_comm-wet.rosinstall`

**Desktop:** ROS, rqt, rviz, and robot-generic libraries

- `$ rosinstall_generator desktop --rosdistro indigo --deps --wet-only --exclude roslisp --tar > indigo-desktop-wet.rosinstall`
- `$ wstool init src indigo-desktop-wet.rosinstall`

This will add all of the `catkin` or `wet` packages in the given variant and then fetch the sources into the `~/ros_catkin_ws/src` directory. The command will take a few minutes to download all of the core ROS packages into the `src` folder. The `-j8` option downloads 8 packages in parallel.

**Packages**: ros/…………………………………………

Some package are excluded because dependency problems as like : ***roslisp*** *p*ackage dependent on ***sbcl.*** Building ***sbcl*** is possible but not tested. However we have two variant namely ***robot*** and ***perception***. If we want just change the package path as like example of ***robot*** below.

```
  $ rosinstall_generator robot --rosdistro indigo --deps --wet-only --
tar > indigo-robot-wet.rosinstall
  $ wstool init src indigo-robot-wet.rosinstall
```

If `wstool init` fails or interrupted, we can resume the download by running.

```
wstool update -t src
```

**What are varients? :** http://www.ros.org/reps/rep-0131.html#variants

Variants are alternatives to removes dependency problems. The releasing Diamond pack of variants is known as REP which gives new entry points for the ROS installation process. Due to continuous development, it is separate from the GUI dependencies resulting lighter weight and GUI-less variants. We define three main entry points for ROS users:

i) Desktop-full(recommended) : for novice user to complete most entry tutorials
ii) Desktop: minimal libraries and tools
iii) ros-base : embedded platforms

Furthermore see this : `http://www.ros.org/reps/rep-0108.html`

## Resolve Dependencies

Before building catkin workspace, we need to make sure that we have all required dependencies. We use ***rosdep*** tool for making this, however some dependencies are not available in the repositories. So have to build manually.

### Unavailable Dependencies

Following packages are not available

For Raspbian Wheezy:

iv) `libconsole-bridge-dev`,
v) `liburdfdom-headers-dev`,
vi) `liburdfdom-dev`, `liblz4-dev`,
vii) `collada-dom-dev`

For Raspbian Jessie:

i) `collada-dom-dev`

Following packages are needed for each ROS variant:

**For Ros_Comm**:

i) `libconsole-bridge-dev`,
ii) `liblz4-dev`

For **Desktop:**

i) `libconsole-bridge-dev`,
ii) `liblz4-dev`,
iii) `liburdfdom-headers-dev`,
iv) `liburdfdom-dev`,
v) `collada-dom-dev`

## Building New directories

The required packages can be built from source in a new directory as making external_src:

- `$ mkdir ~/ros_catkin_ws/external_src`
- `$ sudo apt-get install checkinstall cmake`
- `$ sudo sh -c 'echo "deb-src http://mirrordirector.raspbian.org/raspbian/ testing main contrib non-free rpi" >> /etc/apt/sources.list'`
- `$ sudo apt-get update`

### libconsole-bridge-dev:

- `$ cd ~/ros_catkin_ws/external_src`
- `$ sudo apt-get build-dep console-bridge`
- `$ apt-get source -b console-bridge`

- `$ sudo dpkg -i libconsole-bridge0.2*.deb libconsole-bridge-dev_*.deb`

If you see an error complaining about the option '-std=c++11' then follow these steps to upgrade to gcc 4.7+

- `sudo apt-get install g++-4.7`
- `sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-4.6 60 --slave /usr/bin/g++ g++ /usr/bin/g++-4.6`
- `sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-4.7 40 --slave /usr/bin/g++ g++ /usr/bin/g++-4.7`
- `sudo update-alternatives --config gcc`

**`liblz4-dev`**:

- `$ cd ~/ros_catkin_ws/external_src`
- `$ apt-get source -b lz4`
- `$ sudo dpkg -i liblz4-*.deb`

**`liburdfdom-headers-dev`**:

- `$ cd ~/ros_catkin_ws/external_src`
- `$ git clone https://github.com/ros/urdfdom_headers.git`
- `$ cd urdfdom_headers`
- `$ cmake .`
- `$ sudo checkinstall make install`
    - When check-install asks for any changes, the name (2) needs to change from "urdfdom-headers" to "liburdfdom-headers-dev" otherwise the rosdep install wont find it.

**`liburdfdom-dev`**:

- `$ cd ~/ros_catkin_ws/external_src`
- `$ sudo apt-get install libboost-test-dev libtinyxml-dev`
- `$ git clone https://github.com/ros/urdfdom.git`
- `$ cd urdfdom`
- `$ cmake .`
- `$ sudo checkinstall make install`
    - When check-install asks for any changes, the name (2) needs to change from "urdfdom" to "liburdfdom-dev" otherwise the rosdep install wont find it.

**`collada-dom-dev`**: (Note: You will also need to patch collada_urdf as described here):

- `$ cd ~/ros_catkin_ws/external_src`
- `$ sudo apt-get install libboost-filesystem-dev libxml2-dev`
- `$ wget http://downloads.sourceforge.net/project/collada-dom/Collada%20DOM/Collada%20DOM%202.4/collada-dom-2.4.0.tgz`
- `$ tar -xzf collada-dom-2.4.0.tgz`
- `$ cd collada-dom-2.4.0`
- `$ cmake .`

- `$ sudo checkinstall make install`
    - When check-install asks for any changes, the name (2) needs to change from "collada-dom" to "collada-dom-dev" otherwise the rosdep install wont find it.

**Note:** If you don't want to compile Collada but would like to install the desktop variant, use the following generator:

```
$ rosinstall_generator desktop --rosdistro indigo --deps --wet-only --exclude
roslisp collada_parser collada_urdf --tar > indigo-desktop-wet.rosinstall
```

## Resolving Dependencies with rosdep

The remaining dependencies should be resolved by running ***rosdep:***

*Raspbian Wheezy:*

```
$ cd ~/ros_catkin_ws
$ rosdep install --from-paths src --ignore-src --rosdistro indigo -y -r --
os=debian:wheezy
```

*Raspbian Jessie:*

```
$ cd ~/ros_catkin_ws
$ rosdep install --from-paths src --ignore-src --rosdistro indigo -y -r --
os=debian:jessie
```

Now we are completing all of package in src directory and find all of dependencies they have.

- The `--from-paths src` options says that we want to install the dependencies for an entire directory of packages.
- The `--ignore-src` options says that ignore ROS packages because we are going to install those package manually.
- The `--rosdistro` option indicate building version of ROS so we have to indicate it for environment setup.
- Finally, `-y` options says to ***rosdep*** for less annoying prompts from package manager.

After some time rosdep finish its installing the system dependencies so we can continue but rodesp report some error reports about `python-rosdep`, `python-catkin-pkg`, `python-rospkg`, and `python-rosdistro`. But we can ignore it because they are already install with pip.

## Building the catkin Workspace

After completing downloading those packages we have to resolve the dependencies. Now we are ready to build the catkin packages because we already download it.

**Invoke catkin_make_isolated:**

It is already downloaded and used for installing catkin. We should encountered an internal compiler error due to out of memory. http://raspberrypimaker.com/adding-swap-to-the-raspberrypi/ We should quickly fix that problem by adding swap space to the Pi and recompile. By default the command has –j4 option but we try building by –j2 option. As below

```
  $ sudo ./src/catkin/bin/catkin_make_isolated --install -
DCMAKE_BUILD_TYPE=Release --install-space /opt/ros/indigo -j2
```

We must ensure the ROS file location. Now ROS should be installed! Remember to source the new installation. By defaut in raspberry system it is located at `/opt/ros/indigo` at Ubuntu system . But in raspberry pi it is located at `/ home/user/indigo`.

```
  $ source /opt/ros/indigo/setup.bash or $ source
/home/user/indigo/setup.
```

Or optionally source the `setup.bash` in the `~/.bashrc`, so that ROS environment variables are automatically added to your bash session every time a new shell is launched:

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

## c) Maintaining a Source Checkout

### Updating the workspace

We use same steps for updating source install instructions of Ubuntu to the Raspberry pi. For updating, first we move our ***rosinstall file*** so that it doesn't get overwritten, and generate an updated version. For "desktop-full" variant we use this command line.

```
$ mv -i indigo-desktop-full-wet.rosinstall indigo-desktop-full-
wet.rosinstall.old
$ rosinstall_generator desktop_full --rosdistro indigo --deps --wet-only --
tar > indigo-desktop-full-wet.rosinstall
```

For other variant we can update the filesnames and rosinstall_generator arguments appropriately. After this we compare the new rosinstall file to the old version to see which packages will be updated.

```
$ diff -u indigo-desktop-full-wet.rosinstall indigo-desktop-full-
wet.rosinstall.old
```

If we are satisfied with these changes, incorporate the new rosinstall file into the workspace and update your workspace.

```
$ wstool merge -t src indigo-desktop-full-wet.rosinstall
$ wstool update -t src
```

Now our workspace is up to date with the latest sources. We have to rebuild it.

```
$ ./src/catkin/bin/catkin_make_isolated --install
```

If we specified the `--install-space` option when our workspace initially, we should specify it again when rebuilding our workspace.

Once our workspace has been rebuilt, we should source the setup files again.

```
$ source ~/ros_catkin_ws/install_isolated/setup.bash
```

## Adding Released packages

If we want to add additional packages to the installed ros workspace that have been released into the ros ecosystem. First, a new rosinstall file must be created including the new packages which can be built initially. For example, if we have installed ***ros_comm***, but want to add ***ros_control*** and ***joystick_drivers*** then command will be.

```
  cd ~/ros_catkin_ws
  $ rosinstall_generator ros_comm ros_control joystick_drivers --
rosdistro indigo --deps --wet-only --exclude roslisp --tar > indigo-
custom_ros.rosinstall
```

We can keep listing as many ROS packages as we'd like separated by spaces .

```
  $ wstool update -t src
```

After finishing the workspace, we can run rosdep to install any new dependencies that required as :

***Raspbian Wheezy:***

```
$ rosdep install --from-paths src --ignore-src --rosdistro indigo -y -r --
os=debian:wheezy
```

***Raspbian Jessie:***

```
$ rosdep install --from-paths src --ignore-src --rosdistro indigo -y -r --
os=debian:jessie
```

Finally, now our workspace is up to date and dependencies are satisfied, so we again rebuild the workspace.

```
$ sudo ./src/catkin/bin/catkin_make_isolated --install -
DCMAKE_BUILD_TYPE=Release --install-space /opt/ros/indigo
```

**ROS installation on Ubuntu system:**

The main thing of ROS installation in Ubuntu system and ROS software is matching. There are different version of Ubuntu system and ROS software as like windows operating system 7/8/XP/10. We should be careful about issues of compatible with each other which can be know from ROS/installation site as http://wiki.ros.org/ROS/Installation. We recommend installing latest version of Ubuntu system because maintainer are building more efficient packages which has easier step of installation. The non-profit organization **Open Source Robotics Foundation (OSRF)** built and hosted these packages through online services. Here we discuss ROS installations instructions of ROS kinetic distribution in Ubuntu Xenial (16.04 LTS) platforms.

## Installation

### Configure your Ubuntu repositories

Configure your Ubuntu repositories to allow "restricted," "universe," and "multiverse." You can follow the Ubuntu guide for instructions on doing this.

### Setup your sources.list

Setup your computer to accept software from packages.ros.org. ROS Kinetic **ONLY** supports Wily (Ubuntu 15.10), Xenial (Ubuntu 16.04) and Jessie (Debian 8) for debian packages.

- `sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'`

### Set up your keys

- `sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 0xB01FA116`

### Installation

First, make sure your Debian package index is up-to-date:

- `sudo apt-get update`

There are many different libraries and tools in ROS. We provided four default configurations to get you started. You can also install ROS packages individually.

- **Desktop-Full Install: (Recommended)** : ROS, rqt, rviz, robot-generic libraries, 2D/3D simulators, navigation and 2D/3D perception
  - `sudo apt-get install ros-kinetic-desktop-full`
- **Desktop Install:** ROS, rqt, rviz, and robot-generic libraries
  - `sudo apt-get install ros-kinetic-desktop`

- **ROS-Base: (Bare Bones)** ROS package, build, and communication libraries. No GUI tools.
    - `sudo apt-get install ros-kinetic-ros-base`
- **Individual Package:** You can also install a specific ROS package (replace underscores with dashes of the package name):
    - `sudo apt-get install ros-kinetic-PACKAGE`

      e.g.

      ```
      sudo apt-get install ros-kinetic-slam-gmapping
      ```

To find available packages, use:

```
apt-cache search ros-kinetic
```

## Initialize rosdep

Before you can use ROS, you will need to initialize `rosdep`. `rosdep` enables you to easily install system dependencies for source you want to compile and is required to run some core components in ROS.

```
sudo rosdep init
rosdep update
```

## Getting rosinstall

rosinstall is a frequently used command-line tool in ROS that is distributed separately. It enables you to easily download many source trees for ROS packages with one command.

To install this tool on Ubuntu, run:

```
sudo apt-get install python-rosinstall
```

Chapter 3

# Managing ROS Environment:

During the installation of ROS, we will see that we are prompt to source one of several setup.sh files. It required because ROS relies on the notation of combining spaces using the shell environment which makes developing against different version of ROS.

**Environment variables:**

There are many environment variables that we can set to affect the behavior of ROS. Of these, the most important to understand are `ROS_MASTER_URI`, `ROS_ROOT`, and `ROS_PACKAGE_PATH` as they are commonly used in the system and frequently mentioned in documentation.

Environment variables serve a variety of roles in ROS:

- ***Finding packages***: First and foremost, the `ROS_ROOT` and `ROS_PACKAGE_PATH` enable ROS to locate packages and stacks in the file system. You must also set the `PYTHONPATH` so that the Python interpreter can find ROS libraries.
- ***Effecting a Node runtime***: There are also several ROS environment variables that effect how a Node runs. The `ROS_MASTER_URI` is an important environment variable that tells a Node where the Master is. `ROS_IP` and `ROS_HOSTNAME` affect the network address of a Node and `ROS_NAMESPACE` lets you change its namespace. `ROS_LOG_DIR` lets you set the directory where log files are written. Many of these can be overridden by Remapping Arguments as well, which have precedence over environment variables.
- ***Modifying the build system***: `ROS_BINDEPS_PATH`, `ROS_BOOST_ROOT`, `ROS_PARALLEL_JOBS`, and `ROS_LANG_DISABLE` affect where libraries are found, how they are built, and which ones are built.

**Required ROS Environment variables:**

Most systems will also have `ROS_PACKAGE_PATH` set, but the only required environment variables for ROS are `ROS_ROOT`, `ROS_MASTER_URI`, and `PYTHONPATH`. By default these are automatically set for you by sourcing /opt/ros/ROSDISTRO/setup.bash. (Replace ROSDISTRO with the desired ROS distribution, e.g. indigo.)

1) **ROS_ROOT:**

   `ROS_ROOT` sets the location where the ROS core packages are installed.

```
export ROS_ROOT=/home/user/ros/ros
export PATH=$ROS_ROOT/bin:$PATH
```

2) **ROS_MASTER_URI**

   `ROS_MASTER_URI` is a required setting that tells nodes where they can locate the master. It should be set to the XML-RPC URI of the master. Great care should be taken when using `localhost`, as that can lead to unintended behaviors with remotely launched nodes.

```
export ROS_MASTER_URI=http://mia:11311/
```

**3) PHYTHONPATH**

ROS requires that our `PYTHONPATH` be updated, **even if we don't program in Python!** Many ROS infrastructure tools rely on Python and need access to the roslib package for bootstrapping.

```
export PYTHONPATH=$PYTHONPATH:$ROS_ROOT/core/roslib/src
```

## Additional PATH Environment variables

**1) ROS_PACKAGE_PATH**

`ROS_PACKAGE_PATH` is an optional, but very common environment variable that allows you to add more ROS packages from source to your environment. `ROS_PACKAGE_PATH` can be composed of one or more paths separated by your standard OS path separator (e.g. ':' on Unix-like systems). These *ordered* paths tell the ROS system where to search for more ROS packages. If there are multiple packages of the same name, ROS will choose the one that appears on `ROS_PACKAGE_PATH` *first*.

```
export ROS_PACKAGE_PATH=/home/user/ros/ros-pkg:/another/path
```

Note that each entry in `ROS_PACKAGE_PATH` is searched recursively--all ROS packages below the named path will be found.

With the introduction of catkin, `ROS_PACKAGE_PATH` becomes obsolete, and will be kept only for backwards compatibility with rosbuild packages.

## Tutorial 1

### Creating Catkin workspace and Understanding File system concepts:

**Step 1: Managing Environment:**

```
Commad:  $ printenv | grep ROS
```

**Step 2: Accesses to the ROS commands:**

```
source /opt/ros/<distro>/setup.bash
```

here ***<distro>*** is common name for denoting in command. It must be replace by our installed version of ROS. For example: if we install ROS kinetic version then command be

```
$ source /opt/ros/kinetic/setup.bash
```

We will need to this command on every new shell which have to access the ROS command, unless we add this line to our .bashsrc.

Step3: creating a ROS workspace.

a)Let's create a catkin workspace

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

b) Let's build the workspace:

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

Now we should have 'build' and 'devel' folder which contains several setup files. We start our new setup file by :

```
 source devel/setup.bash
```

**Step 3: Inspecting a package in ROS-tutorials:**

Install a tutorial using command:

```
$ sudo apt-get install ros-<distro>-ros-tutorials
```

Note: ***<distro>*** is replace by version of ROS as ***Kinetic, indigo*** etc.

**Step 4: File system Tools:**

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

**a) Using rospack**

It gives information about packages. For example we cover here only find option of rospack.

```
 Command:  $ rospack find roscpp .
```

We will return on: `/opt/ros/kinetic/share/roscpp` on Ubuntu system.

**b) Using roscd**

***roscd*** is the part of ***rosbash*** It changes the package or a stack directly.

```
$ roscd [locationname[/subdir]]     /
$ roscd roscpp                    / it gives verification by running.
```

```
        $ pwd
```

We should be on

```
YOUR_INSTALL_PATH/share/roscpp
```

It also move to subdirectory of a package or stack. Our ROS_PACKAGE_PATH should contain a list of directories where we have ROS packages separated by colons

**c) roscd log**

It takes us where ROS stores log files. If we have not run any ROS any program yet, it produce error.

```
    $ roscd log
```

**d) using rosls**

rosls is part of rosbash. It allows us to ls directly in a package raher than absolute path.

```
$ rosls [locationname[/subdir]]
```

For example: `$ rosls roscpp_tutorials` we return on: `cmake launch package.xml  srv`

**e) Tab Completion**

It is tedious to type out an entire package name. with the help of two tab and commands we can get full package name.

`$ roscd roscpp_tut<<< now push the TAB key >>>` gives `$ roscd roscpp_tutorials/`

`$ roscd turtles<<< now push the TAB key >>>` gives `$ roscd turtlesim/`

**Step 5: Creating a catkin Package:**

For this we must be in catkin_workspace or move to that that space directory.

a) Make sure we are in catkin work space by below command or we have to calculate.

```
$ cd ~/catkin_ws/src
```

b) Now use the `catkin_create_pkg` script to create a new package called 'beginner_tutorials' which depends on std_msgs, roscpp, and rospy:

```
$ catkin_create_pkg beginner_tutorials std_msgs rospy roscpp
```

c) Package dependencies
   i) First- order dependencies

   The first order dependencies can now reviewed with the ***rospack*** tool.

```
 $ rospack depends1 beginner_tutorials
```

   The package.xml file are stored list of dependicies when running catkin_create _pkg.

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

ii) Second order dependencies

In many cases, a dependency will also have its own dependencies.

```
$ rospack depends1 rospy
```

d) Note: Before creating some catkin Package we need an empty catkin workspace.
Command: ***catkin_create_pkg beginner_tutorial std_msgs rospy roscpp***
Then type command again: ***catkin_make***
Then type command again: ***source devel/setup.bash***

e)

**Tutorial : 2**

# Understanding ROS Nodes:

Overview of Graph concepts:

- Nodes: A node is an executable that uses ROS to communicate with other nodes.
- Messages: ROS data type used when subscribing or publishing to a topic.
- Topics: Nodes can *publish* messages to a topic as well as *subscribe* to a topic to receive messages.
- Master: Name service for ROS (i.e. helps nodes find each other)
- rosout: ROS equivalent of stdout/stderr
- roscore: Master + rosout + parameter server (parameter server will be introduced later)

## Client Libraries

ROS client libraries allow nodes written in different programming languages to communicate:

- **rospy** = python client library
- **roscpp** = c++ client library

**steps of working:**

1) Install lightweight simulator: ROS package

```
$ sudo apt-get install ros-<distro>-ros-tutorials
```

For kinetic

```
$ sudo apt-get install ros-kinetic-ros-tutorials
```

Replace '<distro>' with the name of your ROS distribution (e.g. indigo, jade, kinetic).

2) Roscore is the master of all ros fuction. roscore = ros+core : master (provides name service for ROS) + rosout (stdout/stderr) + parameter server. We have to initialize by typing below command.

```
$ roscore
```

## 3) Using rosnode

rosnode = ros+node : ROS tool to get information about a node. Open up a **new terminal-terminal-2** , and let's use **rosnode** to see what running `roscore` did. `rosnode` displays information about the ROS nodes that are currently running. The `rosnode list` command lists these active nodes:

```
$ rosnode list
```

We see the output is :

```
/rosout
```

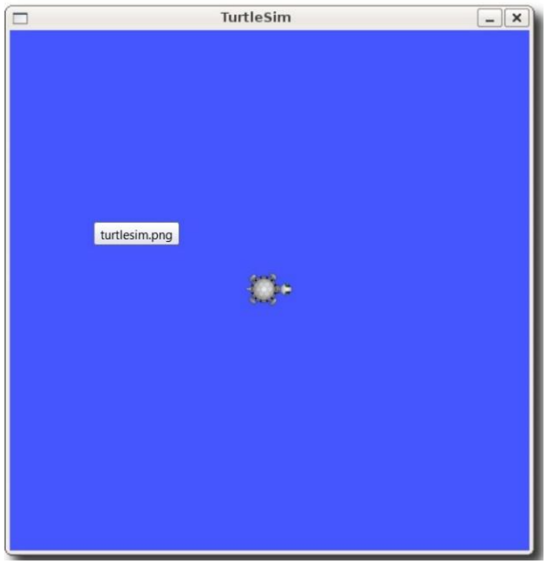
The `rosnode info` command returns information about a specific node.

```
$ rosnode info /rosout
```

## 4) Using rosrun

`rosrun` allows you to use the package name to directly run a node within a package (without having to know the package path). Open up a **new terminal terminal-3**, to run this command ***rosrun*** = ros+run : runs a node from a given package.

```
$ rosrun turtlesim turtlesim_node
```



5) Go to the **terminal 2** and type command:

```
$ rosnode list
```

We see the output is

```
/rosout
/turtlesim
```

6) We can change the turtlesim node's name. For this, close turtlesim window go back to terminal-3 and stop it from running by using `ctrl-C`. The name changing command is commnd is

```
$ rosrun turtlesim turtlesim_node __name:=my_turtle
```

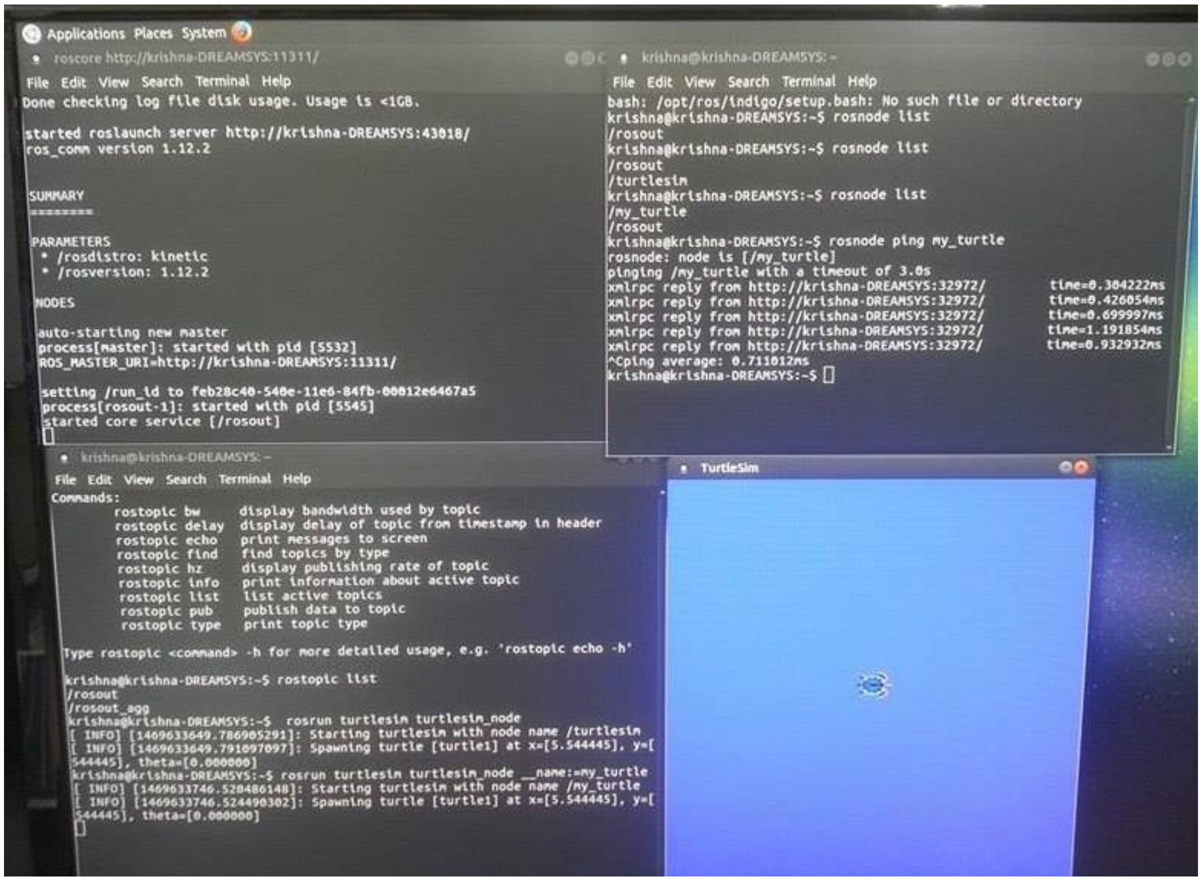
You can check from

`$ rosnode list` Whether or not it is changed.

7) Verification of connection between two terminals by using another rosnode command.

```
$ rosnode ping my_turtle
```

It gives ping time by second wise. Finally your screen is seeing as like below.



Applications Places System
roscore http://krishna-DREAMSYS:11311/
File Edit View Search Terminal Help
krishna@krishna-DREAMSYS: ~
TurtleSim

```
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://krishna-DREAMSYS:43018/
ros_comm version 1.12.2


SUMMARY
========

PARAMETERS
 * /rosdistro: kinetic
 * /rosversion: 1.12.2

NODES

auto-starting new master
process[master]: started with pid [5532]
ROS_MASTER_URI=http://krishna-DREAMSYS:11311/

setting /run_id to feb28c40-540e-11e6-84fb-00012e6467a5
process[rosout-1]: started with pid [5545]
started core service [/rosout]
```

```
bash: /opt/ros/indigo/setup.bash: No such file or directory
krishna@krishna-DREAMSYS:~$ rosnode list
/rosout
krishna@krishna-DREAMSYS:~$ rosnode list
/rosout
/turtlesim
krishna@krishna-DREAMSYS:~$ rosnode list
/my_turtle
/rosout
krishna@krishna-DREAMSYS:~$ rosnode ping my_turtle
rosnode: node is [/my_turtle]
pinging /my_turtle with a timeout of 3.0s
xmlrpc reply from http://krishna-DREAMSYS:32972/      time=0.304222ms
xmlrpc reply from http://krishna-DREAMSYS:32972/      time=0.426054ms
xmlrpc reply from http://krishna-DREAMSYS:32972/      time=0.699997ms
xmlrpc reply from http://krishna-DREAMSYS:32972/      time=1.191854ms
xmlrpc reply from http://krishna-DREAMSYS:32972/      time=0.932932ms
^Cping average: 0.711012ms
krishna@krishna-DREAMSYS:~$
```

```
Commands:
        rostopic bw     display bandwidth used by topic
        rostopic delay  display delay of topic from timestamp in header
        rostopic echo   print messages to screen
        rostopic find   find topics by type
        rostopic hz     display publishing rate of topic
        rostopic info   print information about active topic
        rostopic list   list active topics
        rostopic pub    publish data to topic
        rostopic type   print topic type

Type rostopic <command> -h for more detailed usage, e.g. 'rostopic echo -h'

krishna@krishna-DREAMSYS:~$ rostopic list
/rosout
/rosout_agg
krishna@krishna-DREAMSYS:~$  rosrun turtlesim turtlesim_node
[ INFO] [1469633649.786905291]: Starting turtlesim with node name /turtlesim
[ INFO] [1469633649.791097097]: Spawning turtle [turtle1] at x=[5.544445], y=[
544445], theta=[0.000000]
krishna@krishna-DREAMSYS:~$ rosrun turtlesim turtlesim_node __name:=my_turtle
[ INFO] [1469633746.520486148]: Starting turtlesim with node name /my_turtle
[ INFO] [1469633746.524490302]: Spawning turtle [turtle1] at x=[5.544445], y=[
544445], theta=[0.000000]
```

**Tutorial : 3**

## Understanding ROS Topics

This tutorial introduces ROS topics as well as using the rostopic and rqt_plot commandline tools.

Initial setup:

i) Make sure we have already downloaded file ***ros_tutorials***

***ii)*** ***source*** space ***/opt/ros/kinetic/setup.bash***

***iii)*** ***env | grep ROS***

iv) we can use direct ***source ~/.bashrc***

**Steps :**

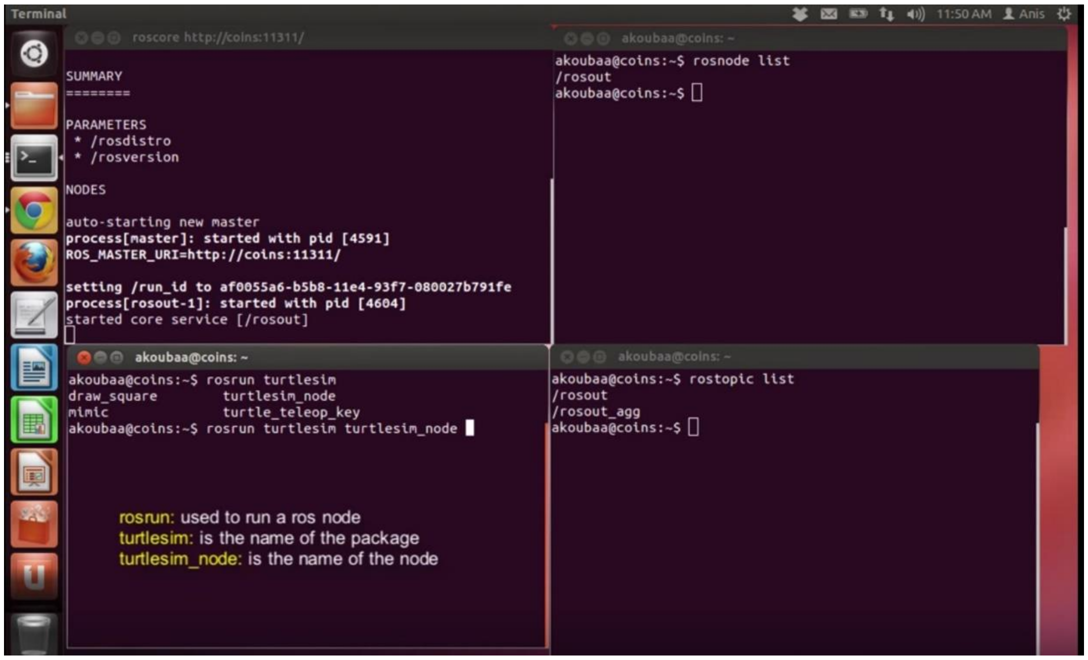
1) **Let's** open a new **terminal-1** for ***roscore*** running. We must ensure only one ***roscore*** is running otherwise it produce error. If it is running please kill before it is launching.

```
$ roscore
```

2) Open a new **terminal-2** for knowing current running rosnodes
   Type Command: `rosnode list`
   it gives list of currently running rosnodes
3) Open a new **terminal-3** for knowing current running rostopics
   Command: `rostopic list`
4) Open a new **terminal-4** for ***turtlesim***.

```
$ rosrun turtlesim turtlesim_node
```

The opening window can be shown as below.



```
SUMMARY
========

PARAMETERS
 * /rosdistro
 * /rosversion

NODES

auto-starting new master
process[master]: started with pid [4591]
ROS_MASTER_URI=http://coins:11311/

setting /run_id to af0055a6-b5b8-11e4-93f7-080027b791fe
process[rosout-1]: started with pid [4604]
started core service [/rosout]
```

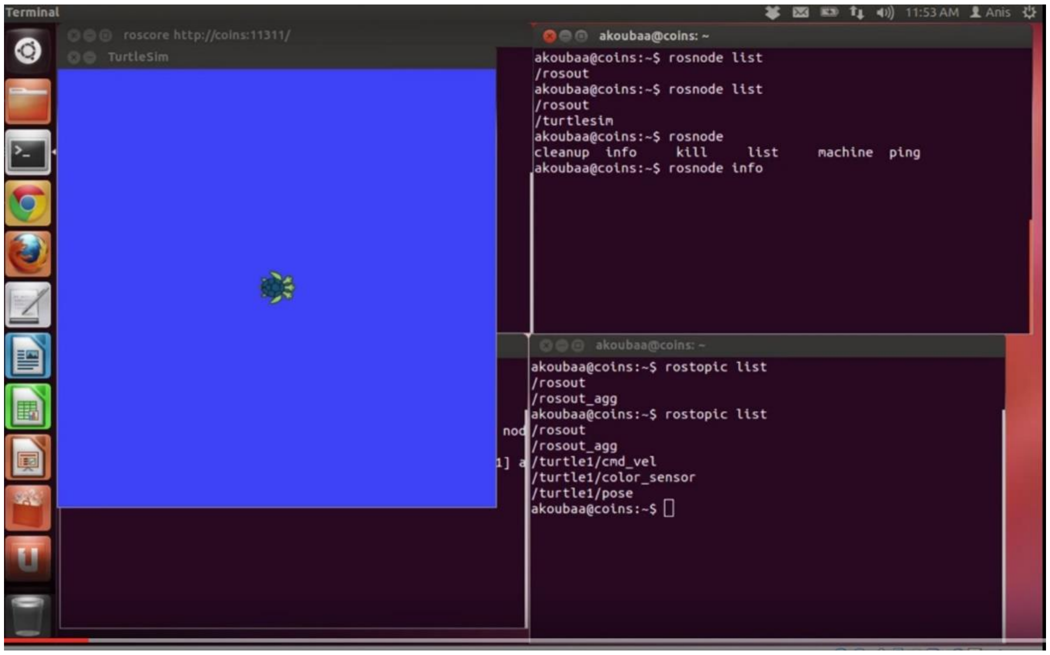
```
akoubaa@coins:~$ rosnode list
/rosout
akoubaa@coins:~$
```

```
akoubaa@coins:~$ rosrun turtlesim
draw_square          turtlesim_node
mimic                turtle_teleop_key
akoubaa@coins:~$ rosrun turtlesim turtlesim_node
```

```
akoubaa@coins:~$ rostopic list
/rosout
/rosout_agg
akoubaa@coins:~$
```

rosrun: used to run a ros node
turtlesim: is the name of the package
turtlesim_node: is the name of the node

5) Now our turtlesim_mode is the one of rosnode. Then our opening window is shown as like below.



```
akoubaa@coins:~$ rosnode list
/rosout
akoubaa@coins:~$ rosnode list
/rosout
/turtlesim
akoubaa@coins:~$ rosnode
cleanup  info     kill     list     machine  ping
akoubaa@coins:~$ rosnode info
```

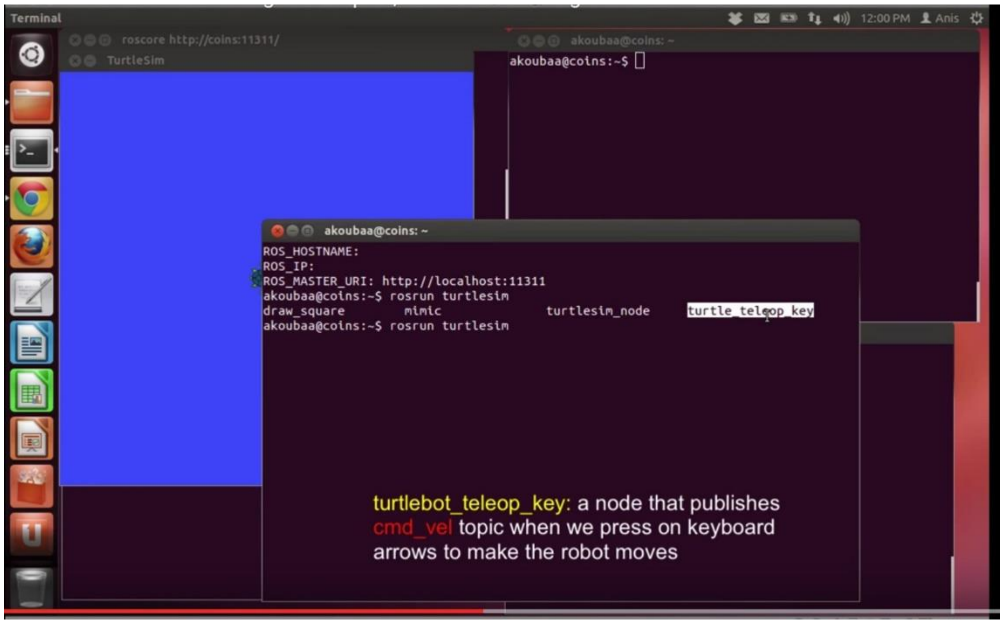
```
akoubaa@coins:~$ rostopic list
/rosout
/rosout_agg
akoubaa@coins:~$ rostopic list
/rosout
/rosout_agg
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
akoubaa@coins:~$
```

Note: we can check again our rosnodes and rostopic list on each corresponding terminal window. Please check one time so we can know each rostopic nodes easily.

6) Turtle keyboard teleoperation
We'll also need something to drive the turtle around with. Please open **in a new terminal-5**:

```
$ rosrun turtlesim turtle_teleop_key
```

Now we can use arrow key of the keyboard to drive turtle around. Figure below shows that our opening window.



```
akoubaa@coins:~$
```

```
ROS_HOSTNAME:
ROS_IP:
ROS_MASTER_URI: http://localhost:11311
akoubaa@coins:~$ rosrun turtlesim
draw_square        mimic            turtlesim_node      turtle_teleop_key
akoubaa@coins:~$ rosrun turtlesim
```

turtlebot_teleop_key: a node that publishes cmd_vel topic when we press on keyboard arrows to make the robot moves

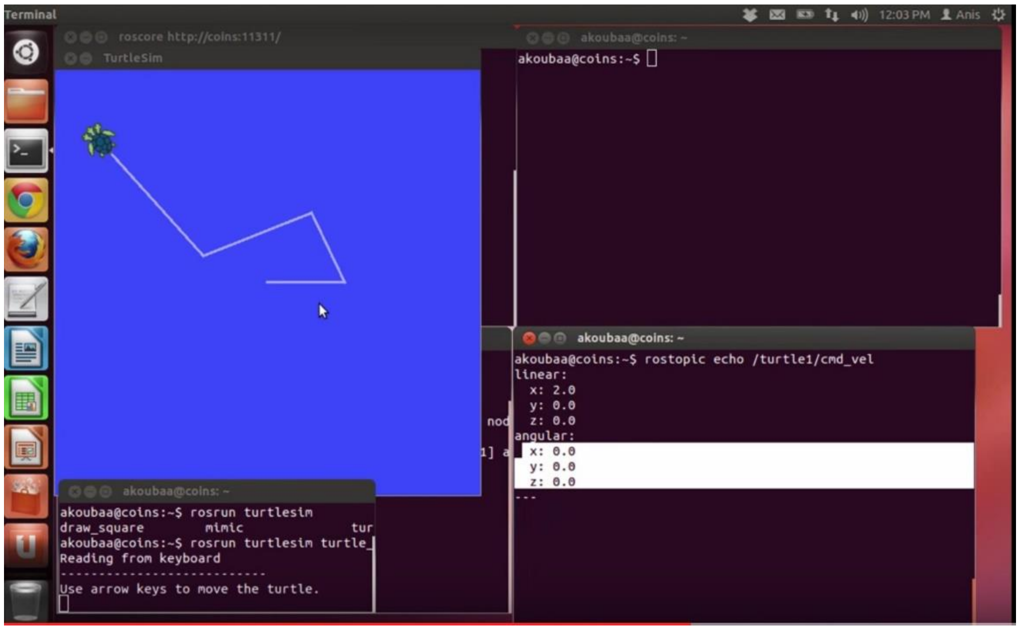
Note: if we use upward downward keys from keyboard then our turtlebot is start to run.

7) Now we are working on **terminal-3** and terminal-5. In terminal- 3 we can see published data by typing command:

```
$ rostopic echo /turtle1/cmd_vel
```

We can see below type of window in our monitor.
For more information about cmd_vel of turtlebot.

Command: `$ rostopic type /turtle1/cmd_vel | rosmsg show`



```
akoubaa@coins:~$ rostopic echo /turtle1/cmd_vel
linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
```

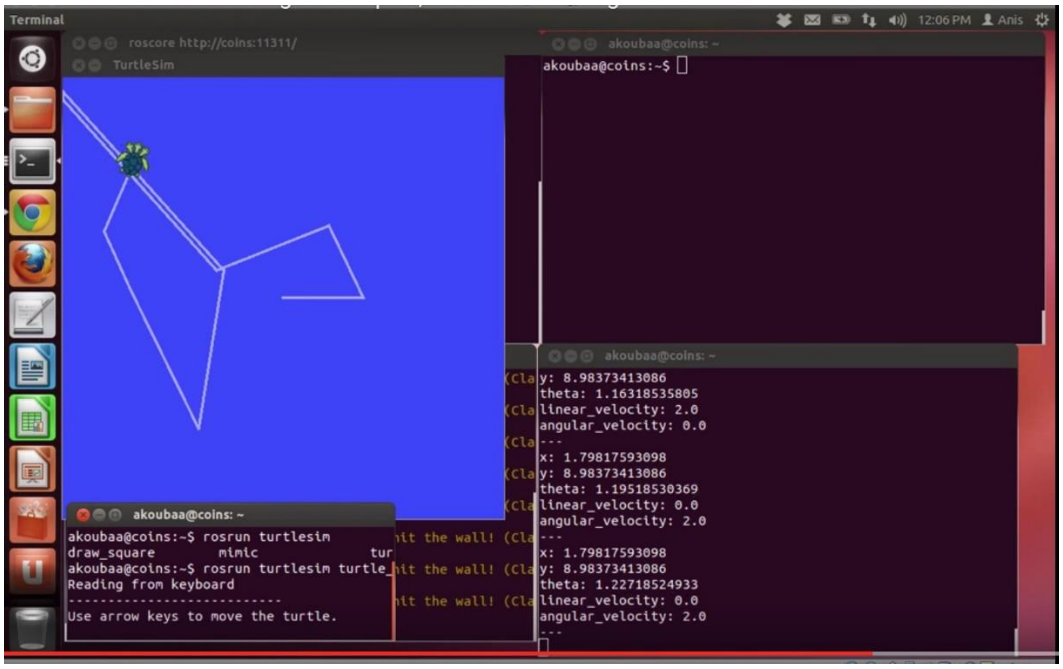
```
akoubaa@coins:~$ rosrun turtlesim
draw_square        mimic              tur
akoubaa@coins:~$ rosrun turtlesim turtle_
Reading from keyboard
---------------------------
Use arrow keys to move the turtle.
```

**Co-ordinate of turtle is (0,0), (0,10) ,(10,0) and (10,10)**

8) Again we are going to find position of turtlebot. For this we type a command in terminal-3
Command:

```
$ rosmsg show geometry_msgs/Twist
```

Command: `$ rostopic type /turtle1/pose`



```
akoubaa@coins:~$
```

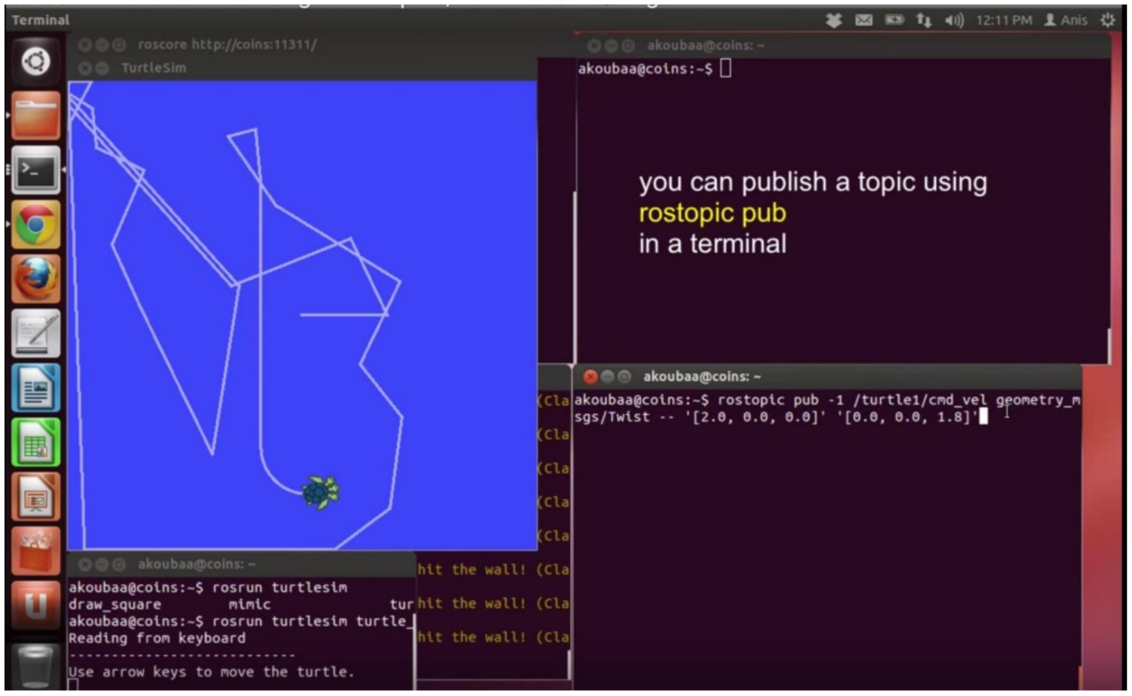
```
y: 8.98373413086
theta: 1.16318535805
linear_velocity: 2.0
angular_velocity: 0.0
---
x: 1.79817593098
y: 8.98373413086
theta: 1.19518530369
linear_velocity: 0.0
angular_velocity: 2.0
---
x: 1.79817593098
y: 8.98373413086
theta: 1.22718524933
linear_velocity: 0.0
angular_velocity: 2.0
---
```

```
akoubaa@coins:~$ rosrun turtlesim
draw_square        mimic             tur
akoubaa@coins:~$ rosrun turtlesim turtle_
Reading from keyboard
---------------------------
Use arrow keys to move the turtle.
```

```
hit the wall! (Cla
hit the wall! (Cla
hit the wall! (Cla
```

9) Pub operation on turtule node provides automatic moving in linear and angular direction. We have command. It published data on to a topic currently advertised

```
  $ rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist -- '[2.0, 0.0,
0.0]' '[0.0, 0.0, 1.8]'
```



```
akoubaa@coins:~$
```

you can publish a topic using
rostopic pub
in a terminal

```
akoubaa@coins:~$ rostopic pub -1 /turtle1/cmd_vel geometry_m
sgs/Twist -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'
```

```
akoubaa@coins:~$ rosrun turtlesim
draw_square        mimic             tur
akoubaa@coins:~$ rosrun turtlesim turtle_
Reading from keyboard
---------------------------
Use arrow keys to move the turtle.
```

```
hit the wall! (Cla
hit the wall! (Cla
hit the wall! (Cla
```

10) We can analysis nodes by seeing its graph. `rqt_graph` creates a dynamic graph of what's going on in the system. rqt_graph is part of the `rqt` package. Unless you already have it installed, run:

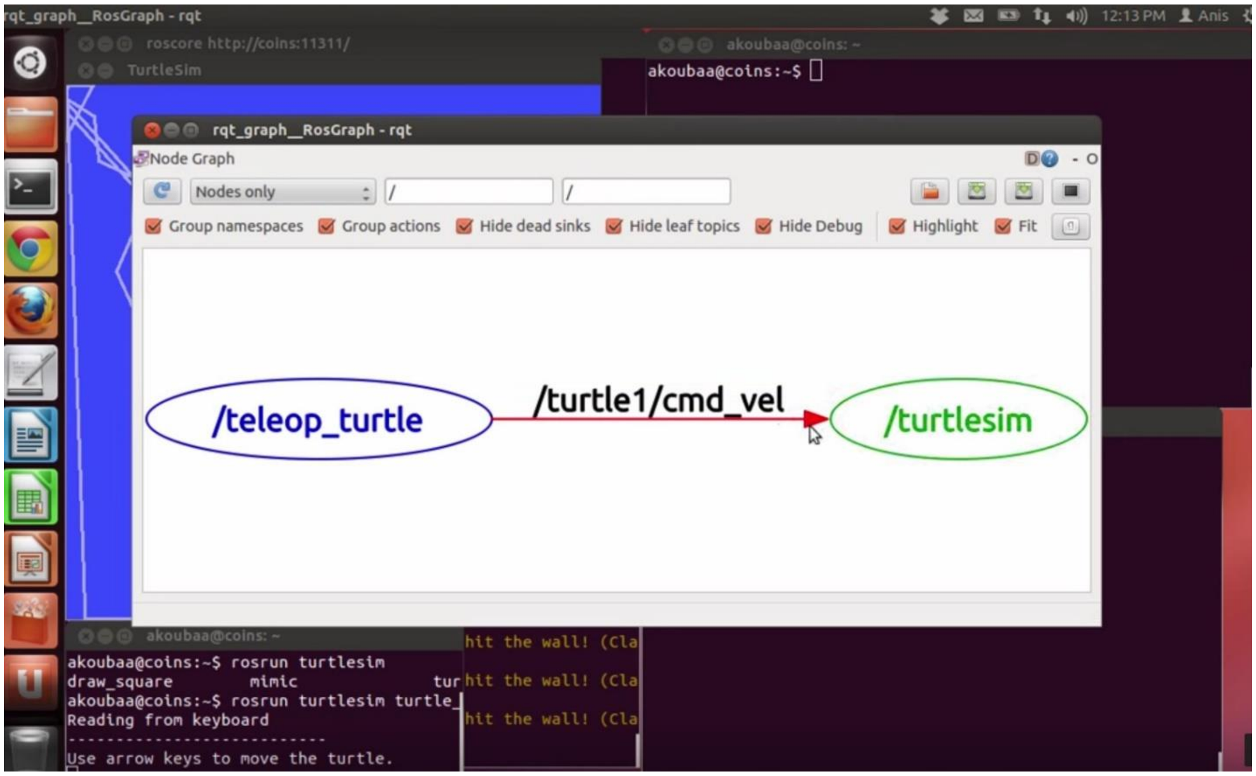
- `$ sudo apt-get install ros-kinetic-rqt`
- `$ sudo apt-get install ros-<distro>-rqt-common-plugins`

replacing <distro> with the name of your ROS distribution (indigo, jade, kinetic)

**In a new terminal**:

```
$ rosrun rqt_graph rqt_graph
```

Our opening window as like below



```
akoubaa@coins:~$
```

```
akoubaa@coins:~$ rosrun turtlesim
draw_square         mimic              tur
akoubaa@coins:~$ rosrun turtlesim turtle_
Reading from keyboard
---------------------------
Use arrow keys to move the turtle.
```

```
hit the wall! (Cla
hit the wall! (Cla
hit the wall! (Cla
```

11) Each time you can refresh this window through refresh icon. Also we can see scrolling time plot by typing command : ***rosrun rqt_plot rqt_plot*** in a new terminal window.

**Comments:**

**Step :1**

**Terminal-1**

a) roscore

**Terminal-2**

b) rosnode list
f) rosnode list
h) rnode info/tuttlesim

**Terminal-4**

d) Rosrun turtlesim+tab+tab
e) Rosrun turtlesim turtlesim_node

**Terminal-3**

c) rostopic list
g) rostopic list
i) rostopic info /turtle1/cmd_vel
j) rosmsg
k) Rosmsg show geometry _msgs/Twist
l) Rrostopic info /turtle1/pose
m) rosmsg show turtlesim/pose

**Step: 2**

**Terminal-1**

Nothing to do

**Terminal-2**

a) rrosnode_list
d) Rosnode info /teleop_turtle

**Terminal-4**

Nothing to do

**Terminal-3**

b) rostopic list
c) rostopic info /turtle1/cmd_vel
f) rostopic echo /turtle1/cmd_vel1
h) Rrostopic echo /turtle1/pose
I) $ rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist – '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'

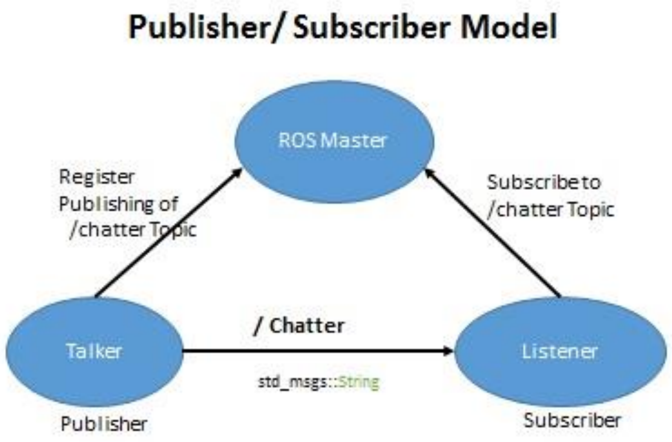
**Terminal-5**

e) Move the robot from arrow key of laptop
f) Press 1, 2 etc.

Chapter 4:

# Publisher and Subscriber In ROS

The message are published by Talker as a publisher by using command ***std_msgs::String*.** The string message are collected and listening by the listener also known as subscriber. They do not communicate with each other in chatter topic.


Publisher/ Subscriber Model
ROS Master
Register Publishing of /chatter Topic
Subscribe to /chatter Topic
Talker
/ Chatter
std_msgs::String
Listener
Publisher
Subscriber

Talker publishes chatter topic and register to the ROS master. Listener will look forward that chatter topic in Master.

**Writing the Publisher Node:**

"Node" is the ROS term for an executable that is connected to the ROS network. Here we'll create a publisher ("talker") node which will continually broadcast a message.

Understanding the code:

`#include "ros/ros.h"` :- Include all header necessary to ROS system
`#include "std_msgs/String.h"` :– Include the ***std_msgs/String*** on automatically generated `String .msg` file in package.
`ros::init(argc, argv, "talker");` :– Initialize the ROS and name of our Node.
`ros::NodeHandle n;` :– handle to the node's process.

`ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);`
Chatter is the topic of publishing string message. Master tell `any nodes to` listen chatter topic which is in publish. Second argument is size of publishing queue. If it is 1000 then maximum buffering message is 1000. `NodeHandle::advertise ()` returns a `ros::Publisher object`, which serves two purposes: 1) it contains a publish messages onto the topic it was created with, and 2) when it goes out of scope, it will automatically unadvertise .

`ros::Rate loop_rate(10);` :- It gives loop duration. It will track and control by `Rate::sleep()`.

In this case we want to run 10hz.

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

`ros::ok()` will return false if:

- a SIGINT is received (Ctrl-C) :- by default due to ***roscpp*** in ROSS
- we have been kicked off the network by another node with the same name
- `ros::shutdown()` has been called by another part of the application.
- all ros::NodeHandles have been destroyed

Once `ros::ok()` returns false, all ROS calls will fail.

Broadcasting of  message:

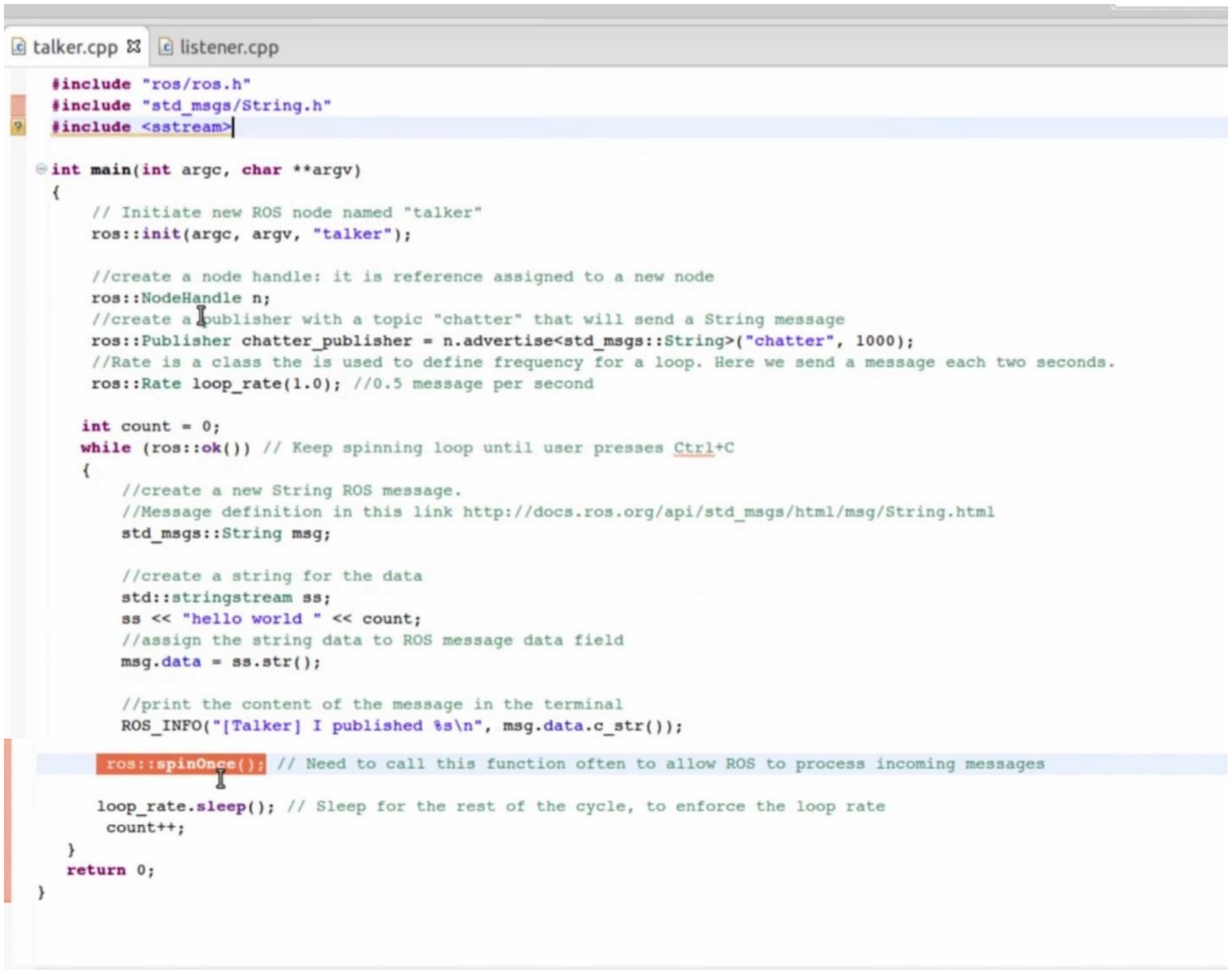
```
  std_msgs::String msg;
  std::stringstream ss;
  ss << "hello world " << count;
   msg.data = ss.str();
```

it is also called message adapted class which generated from msg file. Here “hello world” is standard string message.

`chatter_pub.publish(msg);` :- actual broadcast of data
`ROS_INFO("%s", msg.data.c_str());`:-repelcement of printf/count.
`ros::spinOnce();` :- necessary for larger program where we are receving callbacks.
`loop_rate.sleep();`:-Now we use the ros::Rate object to sleep for the time remaining to let us hit our 10hz publish rate.

**What’s going on in below running program?**

- ➢ Initialize the ROS system
- ➢ Advertise that we are going to be publishing std_msgs/String messages on the `chatter` topic to the master
- ➢ Loop while publishing messages to `chatter` 10 times a second



```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include <sstream>

int main(int argc, char **argv)
{
    // Initiate new ROS node named "talker"
    ros::init(argc, argv, "talker");

    //create a node handle: it is reference assigned to a new node
    ros::NodeHandle n;
    //create a publisher with a topic "chatter" that will send a String message
    ros::Publisher chatter_publisher = n.advertise<std_msgs::String>("chatter", 1000);
    //Rate is a class the is used to define frequency for a loop. Here we send a message each two seconds.
    ros::Rate loop_rate(1.0); //0.5 message per second

   int count = 0;
   while (ros::ok()) // Keep spinning loop until user presses Ctrl+C
   {
       //create a new String ROS message.
       //Message definition in this link http://docs.ros.org/api/std_msgs/html/msg/String.html
       std_msgs::String msg;

       //create a string for the data
       std::stringstream ss;
       ss << "hello world " << count;
       //assign the string data to ROS message data field
       msg.data = ss.str();

       //print the content of the message in the terminal
       ROS_INFO("[Talker] I published %s\n", msg.data.c_str());

      ros::spinOnce(); // Need to call this function often to allow ROS to process incoming messages

      loop_rate.sleep(); // Sleep for the rest of the cycle, to enforce the loop rate
       count++;
  }
  return 0;
}
```

## Writing the Publisher Node:

**Now we** need to receive the messages. The breaking on piece by piece, ignoring some of them is below.

```
 void chatterCallback(const std_msgs::String::ConstPtr& msg)
{ ROS_INFO("I heard: [%s]", msg->data.c_str());
}
```

It is the callback function and getting called when message of ***chatter*** topic arrived. If we want to store we can store otherwise no need to worry about deleting because it passed through ***boost shared_ptr***.

```
ros::Subscriber sub = n.subscribe("chatter", 1000, chatterCallback);
```

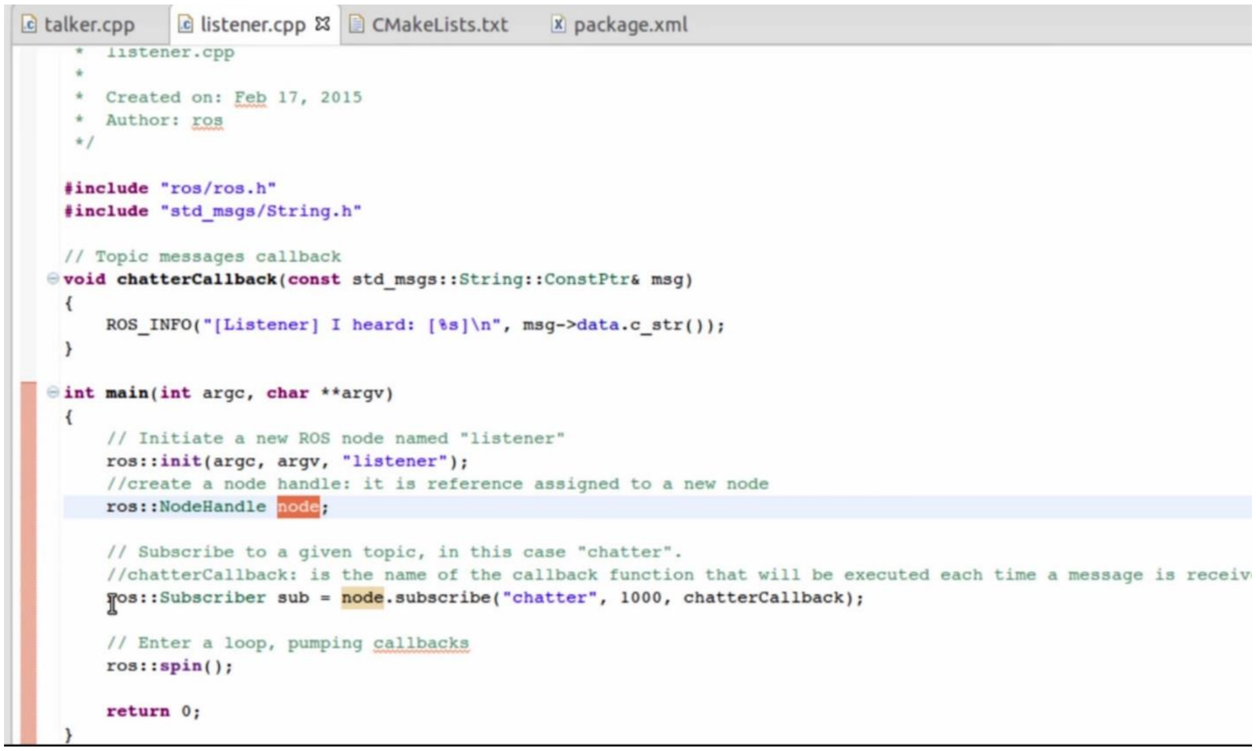
Subscribe to the `chatter` topic with the master. ROS will call the `chatterCallback()` function whenever a new message arrives. If message reach to the equal of queue number 1000, it starts to delete through older messages.

`NodeHandle::subscribe()` returns a `ros::Subscriber` object, that you must hold on to until you want to unsubscribe. When the Subscriber object is destructed, it will automatically unsubscribe from the chatter topic.

`ros::spin();` :-It enter the loop and calling message callbacks as soon as possible. It stops when once `ros::ok()` returns false.

**What's going on in below running program?**

- Initialize the ROS system
- Subscribe to the `chatter` topic
- Spin, waiting for messages to arrive
- When a message arrives, the `chatterCallback()` function is called



```
 * listener.cpp
 *
 * Created on: Feb 17, 2015
 * Author: ros
 */

#include "ros/ros.h"
#include "std_msgs/String.h"

// Topic messages callback
void chatterCallback(const std_msgs::String::ConstPtr& msg)
{
    ROS_INFO("[Listener] I heard: [%s]\n", msg->data.c_str());
}

int main(int argc, char **argv)
{
    // Initiate a new ROS node named "listener"
    ros::init(argc, argv, "listener");
    //create a node handle: it is reference assigned to a new node
    ros::NodeHandle node;

    // Subscribe to a given topic, in this case "chatter".
    //chatterCallback: is the name of the callback function that will be executed each time a message is receiv
    ros::Subscriber sub = node.subscribe("chatter", 1000, chatterCallback);

    // Enter a loop, pumping callbacks
    ros::spin();

    return 0;
}
```

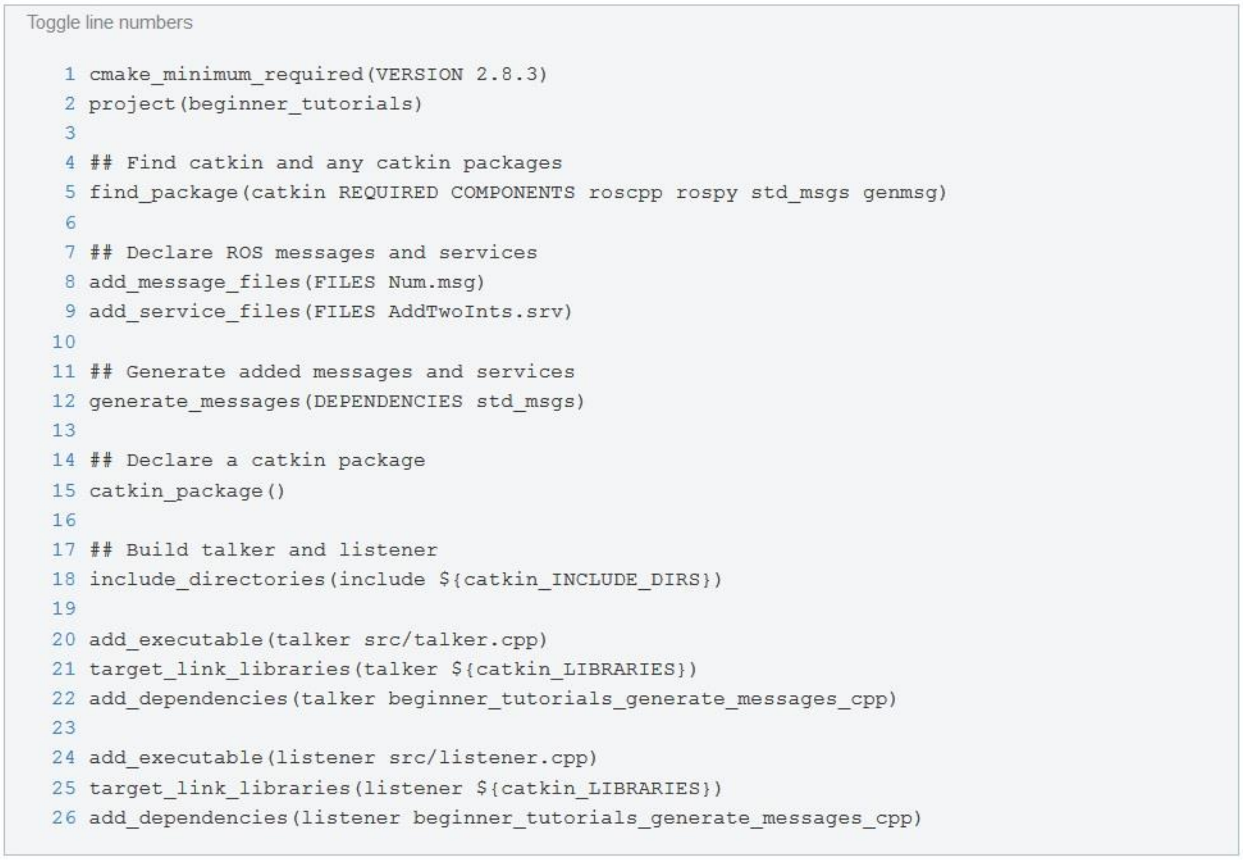
## Building our nodes:

We used catkin_create_pkg in a previous tutorial which created a package.xml and `CMakeLists.txt` for us. Now we are going to create nodes for our publisher-subscriber program by adding some comment(#) in the bottom of your `CMakeLists.txt`.

The adding few lines are listed below.

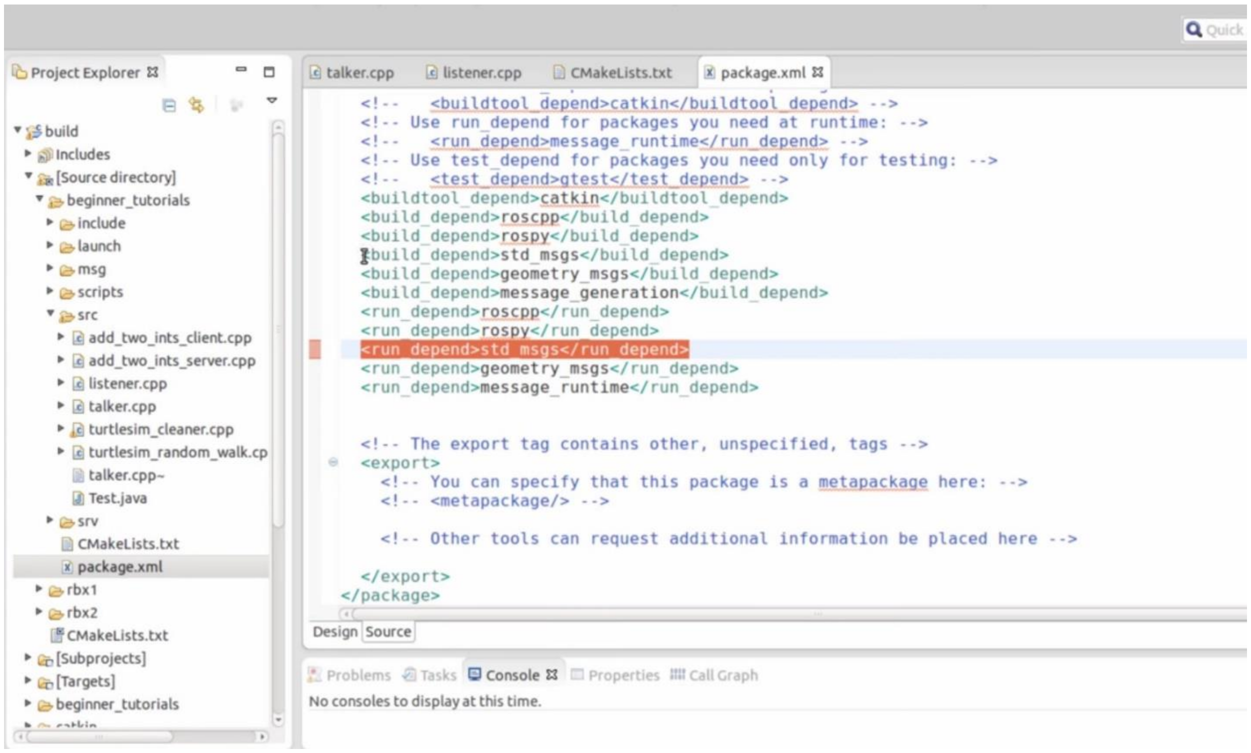
```
include_directories(include ${catkin_INCLUDE_DIRS})

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

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

Our final resulting `CMakeLists.txt` file should look like this:



```
cmake_minimum_required(VERSION 2.8.3)
project(beginner_tutorials)

## Find catkin and any catkin packages
find_package(catkin REQUIRED COMPONENTS roscpp rospy std_msgs genmsg)

## Declare ROS messages and services
add_message_files(FILES Num.msg)
add_service_files(FILES AddTwoInts.srv)

## Generate added messages and services
generate_messages(DEPENDENCIES std_msgs)

## Declare a catkin package
catkin_package()

## Build talker and listener
include_directories(include ${catkin_INCLUDE_DIRS})

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

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

We have to make sure, our ***std_msgs*** dependicies is available for both running and beginning program on package.xml and `CMakeLists.txt.` Making sure by viewing on that files. Figure below shows example of making sure on `package.xml`.



```
  <!--   <buildtool_depend>catkin</buildtool_depend> -->
  <!-- Use run_depend for packages you need at runtime: -->
  <!--   <run_depend>message_runtime</run_depend> -->
  <!-- Use test_depend for packages you need only for testing: -->
  <!--   <test_depend>gtest</test_depend> -->
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>rospy</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_depend>geometry_msgs</build_depend>
  <build_depend>message_generation</build_depend>
  <run_depend>roscpp</run_depend>
  <run_depend>rospy</run_depend>
  <run_depend>std_msgs</run_depend>
  <run_depend>geometry_msgs</run_depend>
  <run_depend>message_runtime</run_depend>


  <!-- The export tag contains other, unspecified, tags -->
  <export>
    <!-- You can specify that this package is a metapackage here: -->
    <!-- <metapackage/> -->

    <!-- Other tools can request additional information be placed here -->

  </export>
</package>
```

```
No consoles to display at this time.
```

**Tutorial 4:**

1) Make a ***beginner_tutorial*** directory within ***catkin_ws/src*** in catkin workspace.
2) Setup environment variables
3) Compile catkin workspace and setup bash file by

    Command: ***catkin_make*** and ***source devel/setup.bash***

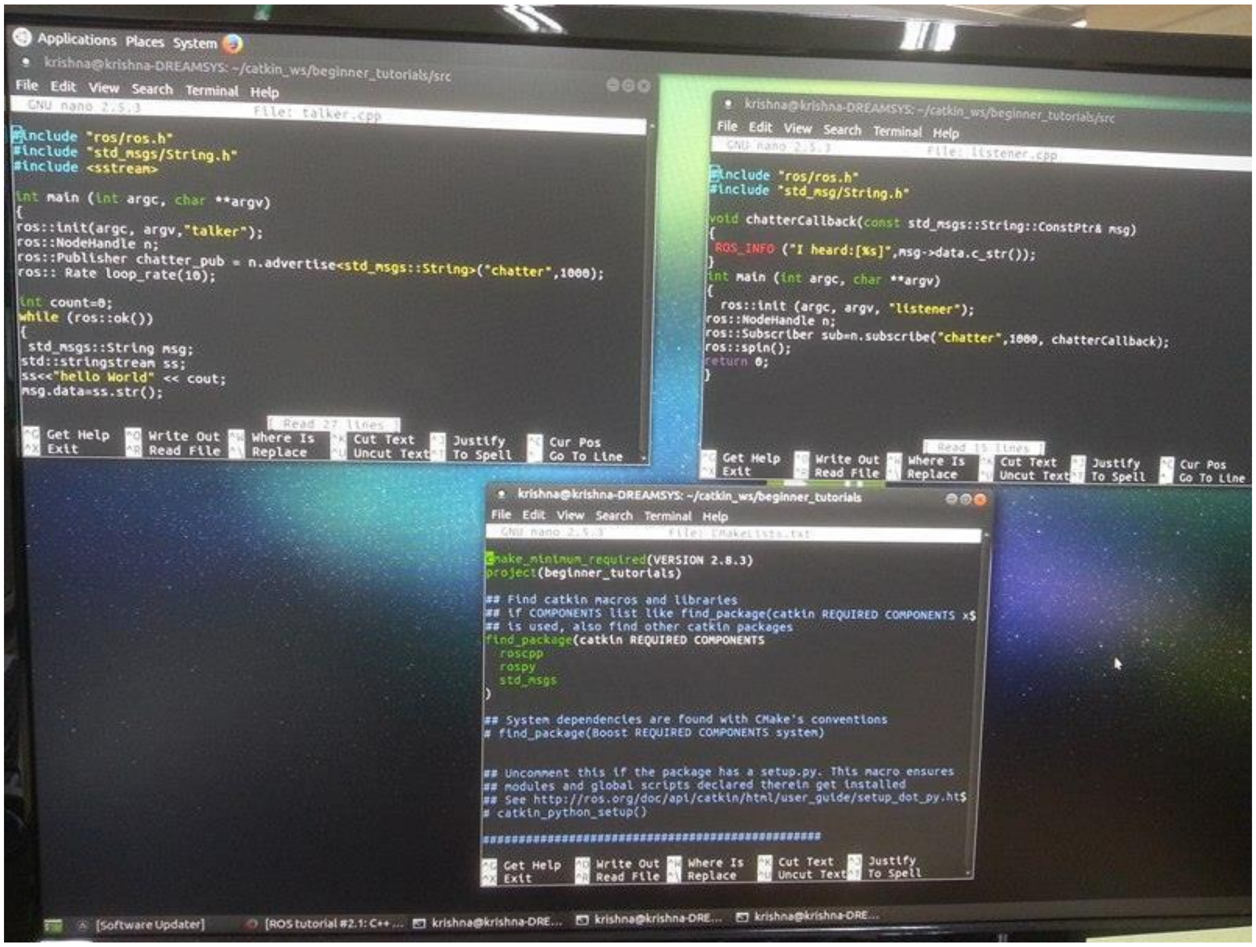
4) In new terminal window-2, type two command for making ***talker.cpp*** and ***listener.cpp*** file inside ***beginner_tutorial/src*** folder.

    Command: ***nano/vim talker.cpp***

    Command: ***nano/vim listener.cpp***

    Note: ***nano*** provide a window where we can write, save and edit the program inside on file.

5) Write a C++ program similar to above or ROS tutorial.



```
krishna@krishna-DREAMSYS: ~/catkin_ws/beginner_tutorials/src
GNU nano 2.5.3                File: talker.cpp

#include "ros/ros.h"
#include "std_msgs/String.h"
#include <sstream>

int main (int argc, char **argv)
{
ros::init(argc, argv,"talker");
ros::NodeHandle n;
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" << cout;
msg.data=ss.str();

                    [ Read 27 lines ]
```

```
krishna@krishna-DREAMSYS: ~/catkin_ws/beginner_tutorials/src
GNU nano 2.5.3                File: listener.cpp

#include "ros/ros.h"
#include "std_msg/String.h"

void chatterCallback(const std_msgs::String::ConstPtr& msg)
{
 ROS_INFO ("I heard:[%s]",msg->data.c_str());
}
int main (int argc, char **argv)
{
  ros::init (argc, argv, "listener");
ros::NodeHandle n;
ros::Subscriber sub=n.subscribe("chatter",1000, chatterCallback);
ros::spin();
return 0;
}

                    [ Read 15 lines ]
```

```
krishna@krishna-DREAMSYS: ~/catkin_ws/beginner_tutorials
GNU nano 2.5.3                File: CMakeLists.txt

cmake_minimum_required(VERSION 2.8.3)
project(beginner_tutorials)

## Find catkin macros and libraries
## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS x$
## is used, also find other catkin packages
find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy
  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.ht$
# catkin_python_setup()

################################################
```

6) Go to ***catkin_ws*** workspace and again compile it by ***catkin_make*** command
7) Go to beginner_tutorial and use nano to Make sure ***std_msgs*** dependencies lies on on ***Package.xml*** and ***CMakeLists.txt***
   Command: ***nano CMakeLists.txt***
8) Edit ***talker.cpp*** as like above.
9) Edit ***listener.cpp*** as like above.
10) Edit CMakeLists.txt as like above.
11) Check all combination.
12) Used to run : ***rosrun cpp_tutorial cpp_tutorial_node***
13)

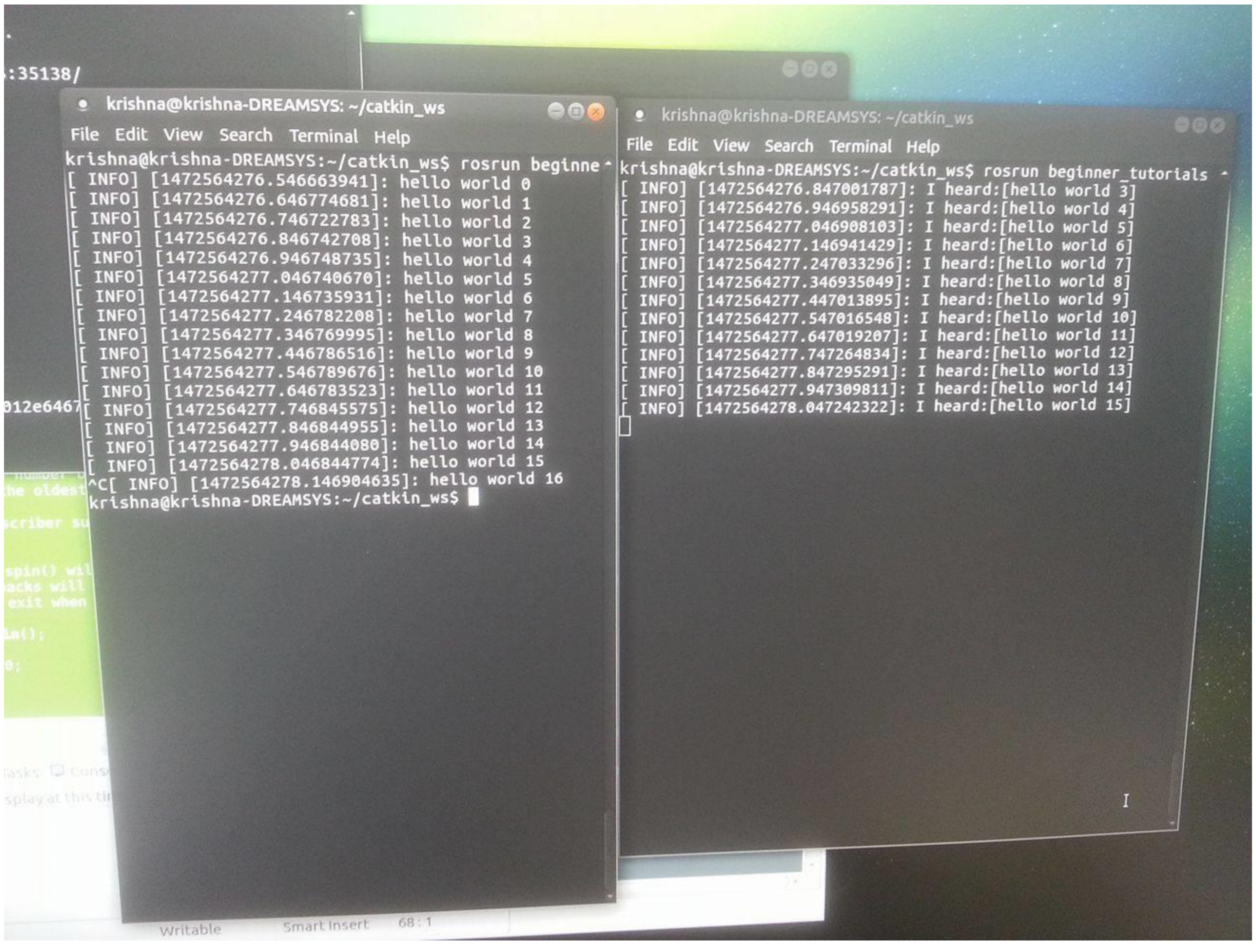
**From eclipse package:**

1) Install eclipse –C/C++ CDT on your Ubuntu packages.
   Command: ***sudo apt-get install eclipse eclipse-cdt g++***

2) Create an eclipse package with in ***beginner_tutorial*** directory.
   Command: ***catkin_create_pkg cpp_tutorial roscpp std_msgs***

Then go to ***cpp_tutorial*** and look packages.

3) Download or create and copy the publisher and subscriber name.
4) Run eclipse by typing command: ***eclipse***
5) Otherwise you can run through desktop.
6) After running eclipse we have to import our build folder of ***catkin_ws.***
7) If we have some basic knowledge about eclipse we can make ***talker.cpp*** and listener.cpp source file easily within ***catkin_ws/src/beginner_tutorial***.
8) After making those file open mate terminal and run ***catkin_make*** command within ***catkin_ws*** workspace.
9) Then type ***roscore*** command. Now our master is running.
10) Then take terminal-2 and type ***rosrun beginnger_tutorials talker***
11) If it works we can see printing message as published by talker
12) Otherwise, we have to run setup file from ROS repository.
13) To run setup file of catkin workspace, we have to type on terminal-2 as: ***Source devel/setup.bash***
14) Then again we can type ***rosrun beginnger_tutorials talker .***
15) Similar process we can do for listener.cpp. But listener does not produces any message unless talker produce.



```
krishna@krishna-DREAMSYS:~/catkin_ws$ rosrun beginne
[ INFO] [1472564276.546663941]: hello world 0
[ INFO] [1472564276.646774681]: hello world 1
[ INFO] [1472564276.746722783]: hello world 2
[ INFO] [1472564276.846742708]: hello world 3
[ INFO] [1472564276.946748735]: hello world 4
[ INFO] [1472564277.046740670]: hello world 5
[ INFO] [1472564277.146735931]: hello world 6
[ INFO] [1472564277.246782208]: hello world 7
[ INFO] [1472564277.346769995]: hello world 8
[ INFO] [1472564277.446786516]: hello world 9
[ INFO] [1472564277.546789676]: hello world 10
[ INFO] [1472564277.646783523]: hello world 11
[ INFO] [1472564277.746845575]: hello world 12
[ INFO] [1472564277.846844955]: hello world 13
[ INFO] [1472564277.946844080]: hello world 14
[ INFO] [1472564278.046844774]: hello world 15
^C[ INFO] [1472564278.146904635]: hello world 16
krishna@krishna-DREAMSYS:~/catkin_ws$
```

```
krishna@krishna-DREAMSYS:~/catkin_ws$ rosrun beginner_tutorials
[ INFO] [1472564276.847001787]: I heard:[hello world 3]
[ INFO] [1472564276.946958291]: I heard:[hello world 4]
[ INFO] [1472564277.046908103]: I heard:[hello world 5]
[ INFO] [1472564277.146941429]: I heard:[hello world 6]
[ INFO] [1472564277.247033296]: I heard:[hello world 7]
[ INFO] [1472564277.346935049]: I heard:[hello world 8]
[ INFO] [1472564277.447013895]: I heard:[hello world 9]
[ INFO] [1472564277.547016548]: I heard:[hello world 10]
[ INFO] [1472564277.647019207]: I heard:[hello world 11]
[ INFO] [1472564277.747264834]: I heard:[hello world 12]
[ INFO] [1472564277.847295291]: I heard:[hello world 13]
[ INFO] [1472564277.947309811]: I heard:[hello world 14]
[ INFO] [1472564278.047242322]: I heard:[hello world 15]
```

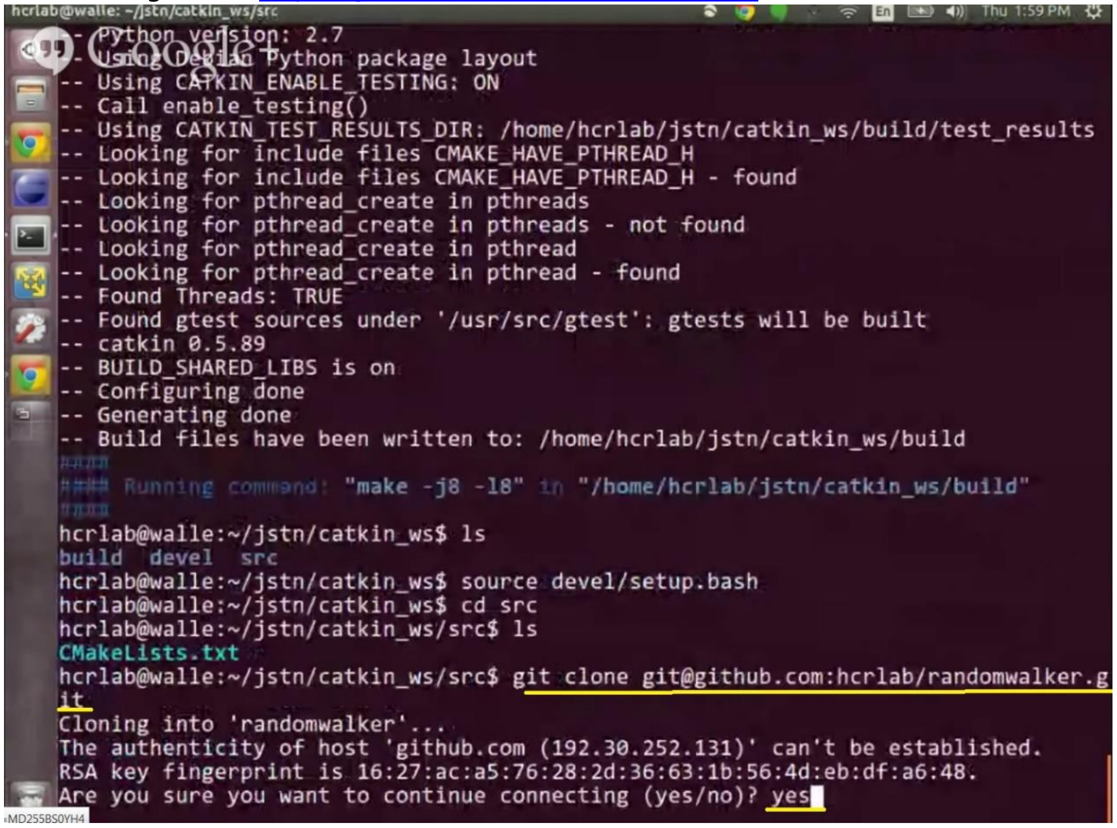
Chapter 5:

# Working with internet Package.

In this chapter we learning about the download repositories of internet packages of ROS and working with them. www.github.com is one of common ROS tutorials packages site. In terminal window, we use some command to download and process it. Let download a ***randomwalker*** repositories with in our catkin_ws workspace.

Package downloading step:

1) Go to package downloading site such as www.github.com/hcrlab.
2) Select an available packages: for example ***randomwalker***.
3) In a terminal window use command
   Command: ***git clone*** https://github.com/hcrlab/randomwalker

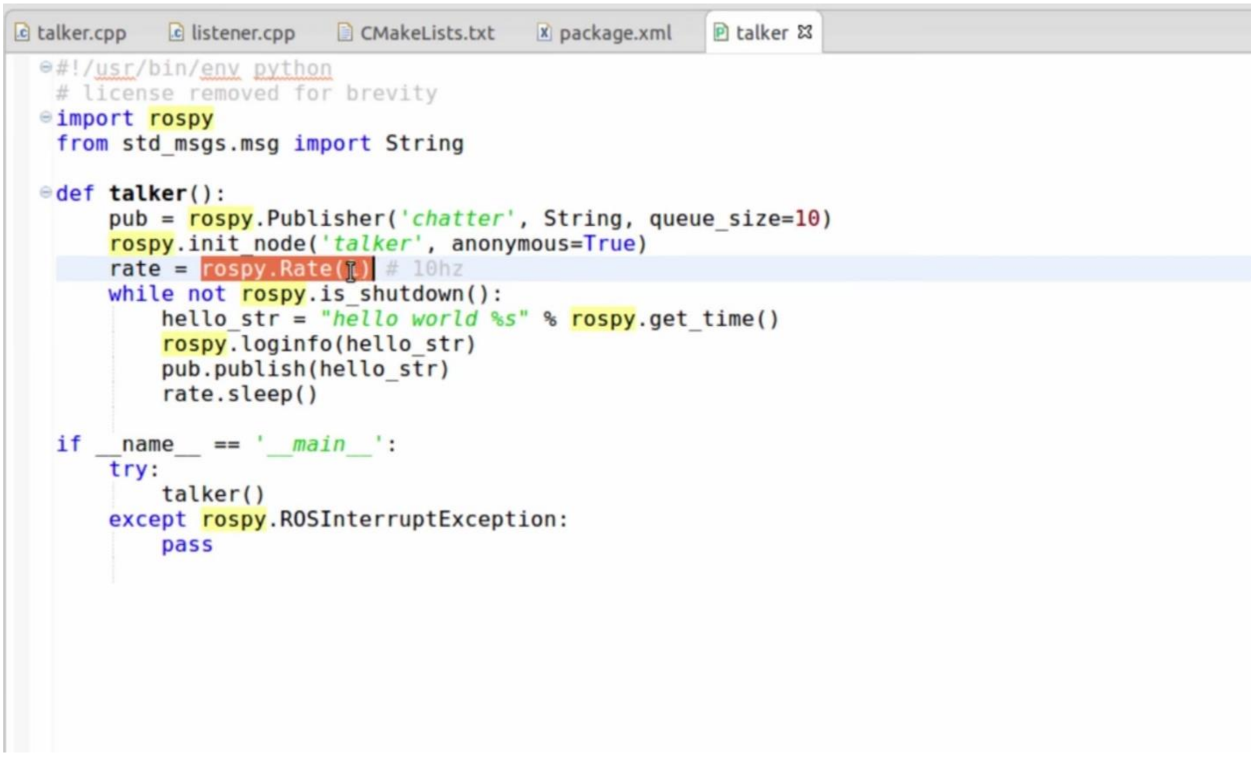


```
hcrlab@walle: ~/jstn/catkin_ws/src
-- Python version: 2.7
-- Using Debian Python package layout
-- Using CATKIN_ENABLE_TESTING: ON
-- Call enable_testing()
-- Using CATKIN_TEST_RESULTS_DIR: /home/hcrlab/jstn/catkin_ws/build/test_results
-- Looking for include files CMAKE_HAVE_PTHREAD_H
-- Looking for include files CMAKE_HAVE_PTHREAD_H - found
-- Looking for pthread_create in pthreads
-- Looking for pthread_create in pthreads - not found
-- Looking for pthread_create in pthread
-- Looking for pthread_create in pthread - found
-- Found Threads: TRUE
-- Found gtest sources under '/usr/src/gtest': gtests will be built
-- catkin 0.5.89
-- BUILD_SHARED_LIBS is on
-- Configuring done
-- Generating done
-- Build files have been written to: /home/hcrlab/jstn/catkin_ws/build
####
#### Running command: "make -j8 -l8" in "/home/hcrlab/jstn/catkin_ws/build"
####
hcrlab@walle:~/jstn/catkin_ws$ ls
build  devel  src
hcrlab@walle:~/jstn/catkin_ws$ source devel/setup.bash
hcrlab@walle:~/jstn/catkin_ws$ cd src
hcrlab@walle:~/jstn/catkin_ws/src$ ls
CMakeLists.txt
hcrlab@walle:~/jstn/catkin_ws/src$ git clone git@github.com:hcrlab/randomwalker.g
it
Cloning into 'randomwalker'...
The authenticity of host 'github.com (192.30.252.131)' can't be established.
RSA key fingerprint is 16:27:ac:a5:76:28:2d:36:63:1b:56:4d:eb:df:a6:48.
Are you sure you want to continue connecting (yes/no)? yes
```

4)
5)

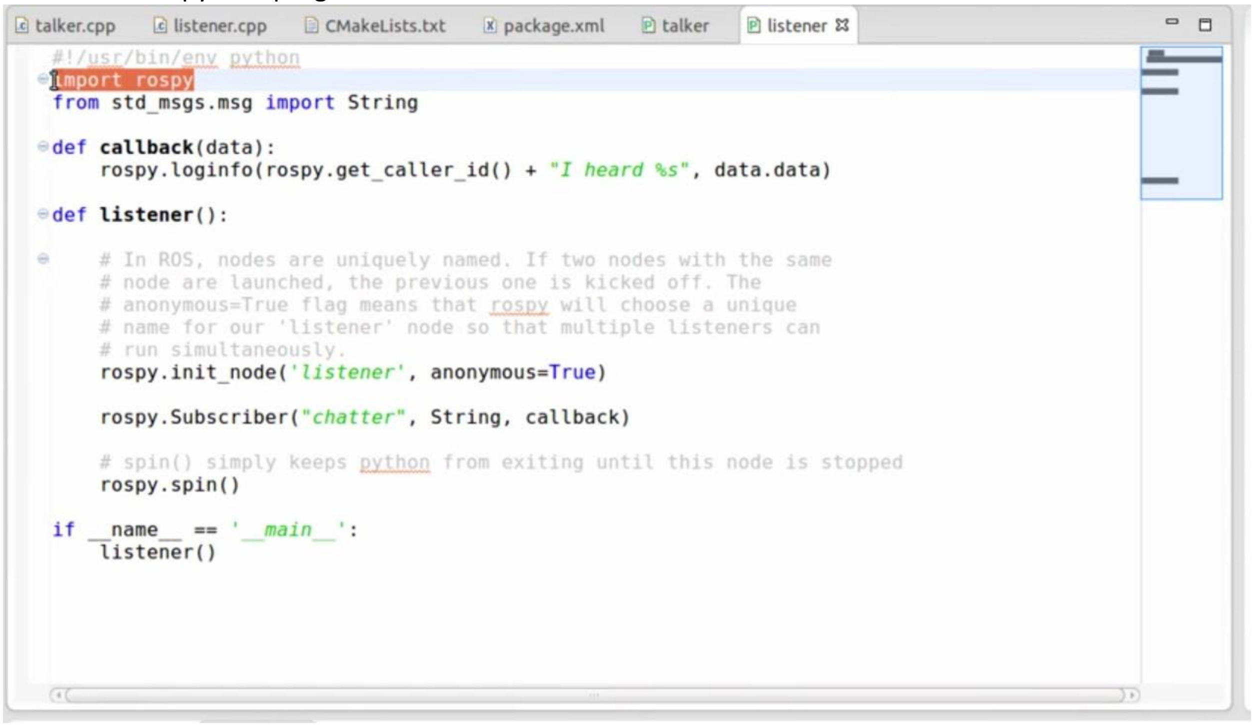
**Publisher with Python:**

The talker of python program looks like as


talker.cpp | listener.cpp | CMakeLists.txt | package.xml | talker

```
#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import String

def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(1) # 10hz
    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

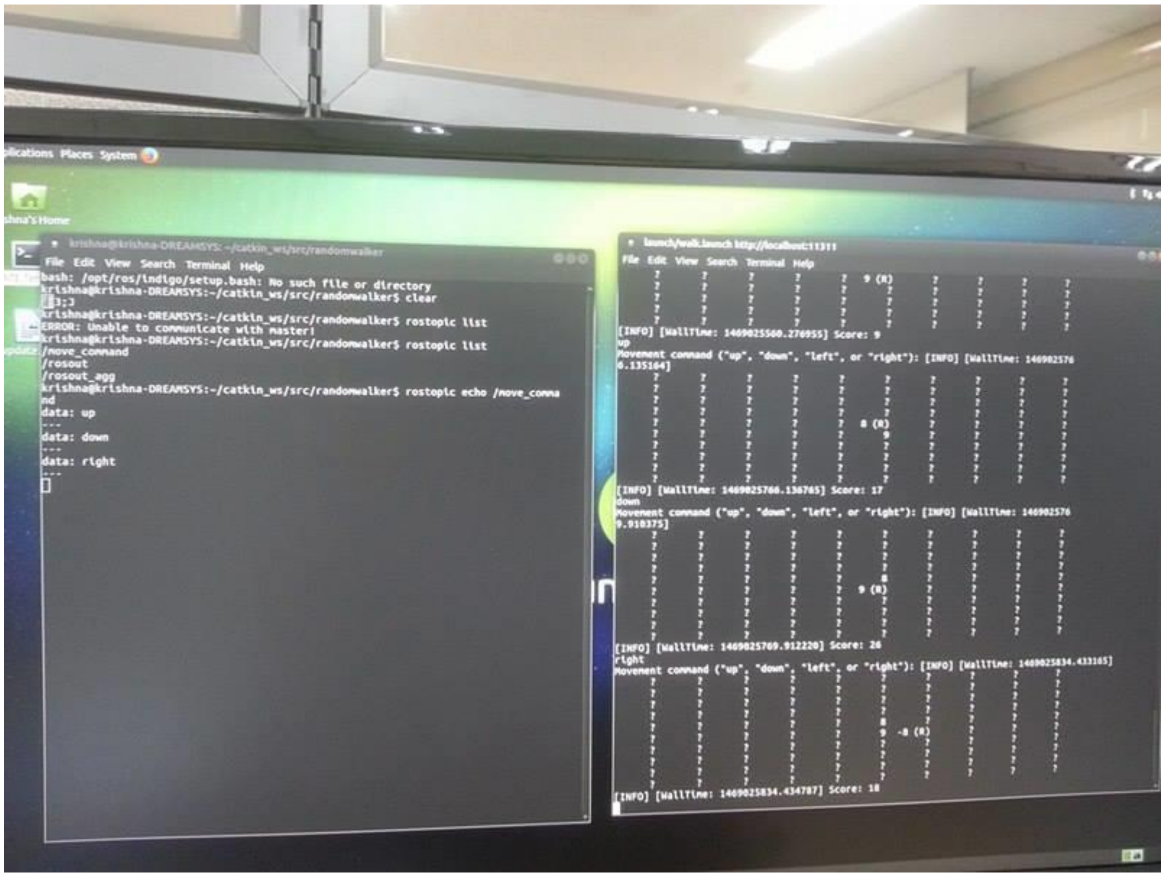
Subscriber with  Python:
The listener of python program look like as:


talker.cpp | listener.cpp | CMakeLists.txt | package.xml | talker | listener

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import String

def callback(data):
    rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)

def listener():

    # In ROS, nodes are uniquely named. If two nodes with the same
    # node are launched, the previous one is kicked off. The
    # anonymous=True flag means that rospy will choose a unique
    # name for our 'listener' node so that multiple listeners can
    # run simultaneously.
    rospy.init_node('listener', anonymous=True)

    rospy.Subscriber("chatter", String, callback)

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()

if __name__ == '__main__':
    listener()
```

**Steps of Tutorial: 5**

1) Go to the ***catkin_ws*** workspace and type ***catkin_make***
2) Setup bash file by typing ***source devel/setup.bash***
3) Go to ***src*** folder and cloning random walker from copying random walker ***url***
4) Go to the random walker folder and type command
   Command: ***git checkout origin/finished***
5) Check downloaded file by ***ls*** command then launch ***walk.launch*** file
   Command: ***roslaunch launch/walk.launch***
6) Open a ***new terminal window-2*** from file menu and type ***rostopic*** command
7) In terminal ***window -2*** type command
   Command: ***rostopic echo /move_command***
8) We can see publishing up , down etc. command on second terminal as shown below

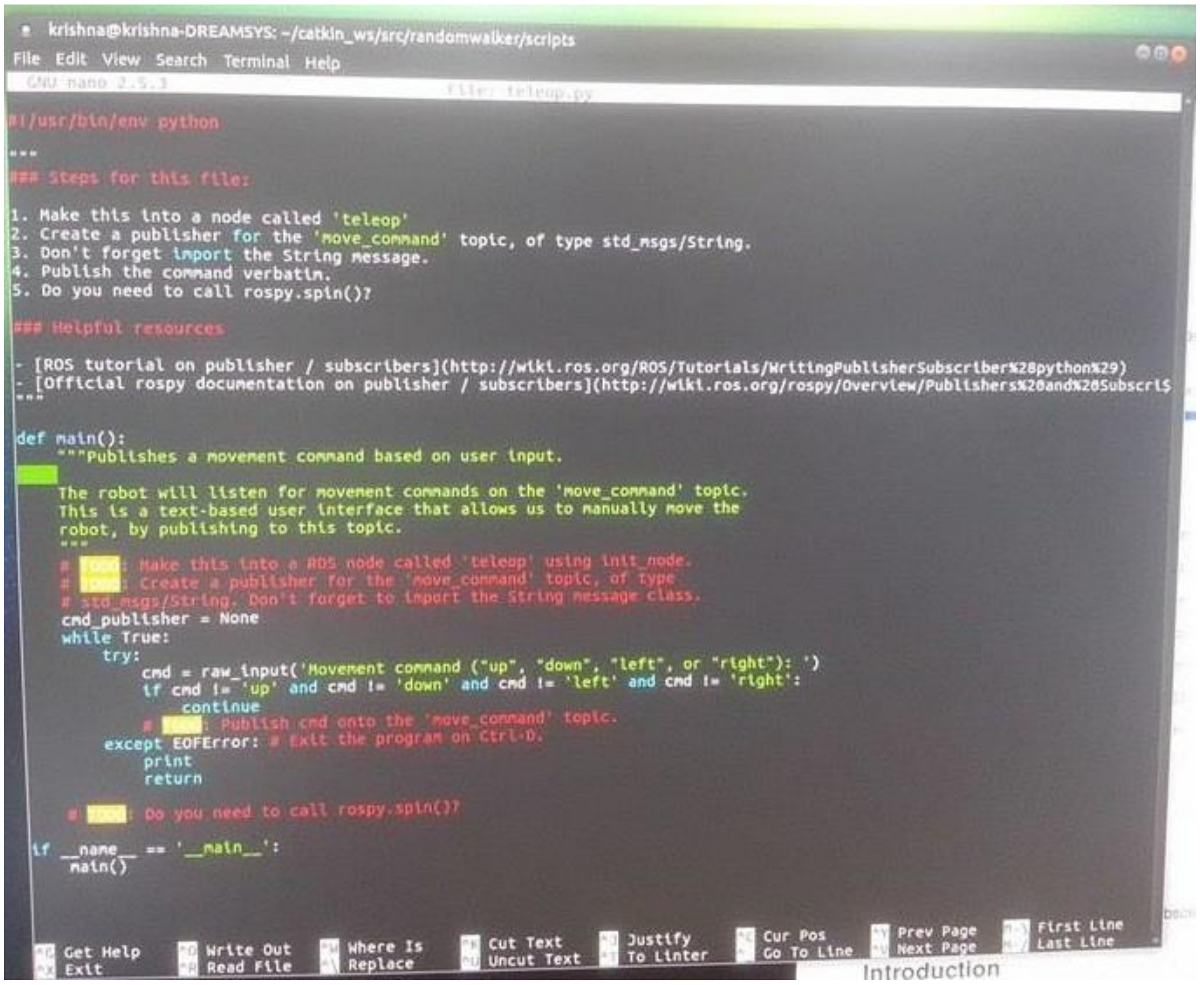

Applications Places System
krishna's Home
krishna@krishna-DREAMSYS: ~/catkin_ws/src/randomwalker
File Edit View Search Terminal Help
launch/walk.launch http://localhost:11311
File Edit View Search Terminal Help

```
bash: /opt/ros/indigo/setup.bash: No such file or directory
krishna@krishna-DREAMSYS:~/catkin_ws/src/randomwalker$ clear
3;J
krishna@krishna-DREAMSYS:~/catkin_ws/src/randomwalker$ rostopic list
ERROR: Unable to communicate with master!
krishna@krishna-DREAMSYS:~/catkin_ws/src/randomwalker$ rostopic list
/move_command
/rosout
/rosout_agg
krishna@krishna-DREAMSYS:~/catkin_ws/src/randomwalker$ rostopic echo /move_comma
nd
data: up
---
data: down
---
data: right
---
```

```
       ?       ?       ?       ?       ?   9 (R)       ?       ?       ?       ?
       ?       ?       ?       ?       ?       ?       ?       ?       ?       ?
       ?       ?       ?       ?       ?       ?       ?       ?       ?       ?
       ?       ?       ?       ?       ?       ?       ?       ?       ?       ?
       ?       ?       ?       ?       ?       ?       ?       ?       ?       ?
[INFO] [WallTime: 1469025560.276955] Score: 9
up
Movement command ("up", "down", "left", or "right"): [INFO] [WallTime: 146902576
6.135164]
       ?       ?       ?       ?       ?       ?       ?       ?       ?       ?
       ?       ?       ?       ?       ?       ?       ?       ?       ?       ?
       ?       ?       ?       ?       ?       ?       ?       ?       ?       ?
       ?       ?       ?       ?       ?       ?       ?       ?       ?       ?
       ?       ?       ?       ?       ?   8 (R)       ?       ?       ?       ?
       ?       ?       ?       ?       ?       9       ?       ?       ?       ?
       ?       ?       ?       ?       ?       ?       ?       ?       ?       ?
       ?       ?       ?       ?       ?       ?       ?       ?       ?       ?
       ?       ?       ?       ?       ?       ?       ?       ?       ?       ?
       ?       ?       ?       ?       ?       ?       ?       ?       ?       ?
[INFO] [WallTime: 1469025766.136765] Score: 17
down
Movement command ("up", "down", "left", or "right"): [INFO] [WallTime: 146902576
9.910375]
       ?       ?       ?       ?       ?       ?       ?       ?       ?       ?
       ?       ?       ?       ?       ?       ?       ?       ?       ?       ?
       ?       ?       ?       ?       ?       ?       ?       ?       ?       ?
       ?       ?       ?       ?       ?       ?       ?       ?       ?       ?
       ?       ?       ?       ?       ?       8       ?       ?       ?       ?
       ?       ?       ?       ?       ?   9 (R)       ?       ?       ?       ?
       ?       ?       ?       ?       ?       ?       ?       ?       ?       ?
       ?       ?       ?       ?       ?       ?       ?       ?       ?       ?
       ?       ?       ?       ?       ?       ?       ?       ?       ?       ?
       ?       ?       ?       ?       ?       ?       ?       ?       ?       ?
[INFO] [WallTime: 1469025769.912220] Score: 26
right
Movement command ("up", "down", "left", or "right"): [INFO] [WallTime: 1469025834.433165]
       ?       ?       ?       ?       ?       ?       ?       ?       ?       ?
       ?       ?       ?       ?       ?       ?       ?       ?       ?       ?
       ?       ?       ?       ?       ?       ?       ?       ?       ?       ?
       ?       ?       ?       ?       ?       ?       ?       ?       ?       ?
       ?       ?       ?       ?       ?       8       ?       ?       ?       ?
       ?       ?       ?       ?       ?       9  -8 (R)       ?       ?       ?
       ?       ?       ?       ?       ?       ?       ?       ?       ?       ?
       ?       ?       ?       ?       ?       ?       ?       ?       ?       ?
       ?       ?       ?       ?       ?       ?       ?       ?       ?       ?
       ?       ?       ?       ?       ?       ?       ?       ?       ?       ?
[INFO] [WallTime: 1469025834.434787] Score: 18
```

9) Turn off the launching by control-c command
10) Type command for checking master
    Command: ***git checkout master***
11) To run python program type
    Command: ***python teleop.py***
    Now we can see again provide input to the program. But this time no publishing

12) Open ***teleop*** file by typing command
    Command: ***nano teleop.py*** or ***vim teleop.py***
    A new window is open for editing those python file. Edit as similar way of python tutorial as shown above.



krishna@krishna-DREAMSYS: ~/catkin_ws/src/randomwalker/scripts
File Edit View Search Terminal Help
GNU nano 2.5.3
File: teleop.py

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

"""
### Steps for this file:

1. Make this into a node called 'teleop'
2. Create a publisher for the 'move_command' topic, of type std_msgs/String.
3. Don't forget import the String message.
4. Publish the command verbatim.
5. Do you need to call rospy.spin()?

### Helpful resources

- [ROS tutorial on publisher / subscribers](http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29)
- [Official rospy documentation on publisher / subscribers](http://wiki.ros.org/rospy/Overview/Publishers%20and%20Subscri$
"""

def main():
    """Publishes a movement command based on user input.

    The robot will listen for movement commands on the 'move_command' topic.
    This is a text-based user interface that allows us to manually move the
    robot, by publishing to this topic.
    """
    # TODO: Make this into a ROS node called 'teleop' using init_node.
    # TODO: Create a publisher for the 'move_command' topic, of type
    # std_msgs/String. Don't forget to import the String message class.
    cmd_publisher = None
    while True:
        try:
            cmd = raw_input('Movement command ("up", "down", "left", or "right"): ')
            if cmd != 'up' and cmd != 'down' and cmd != 'left' and cmd != 'right':
                continue
            # TODO: Publish cmd onto the 'move_command' topic.
        except EOFError: # Exit the program on Ctrl-D.
            print
            return

    # TODO: Do you need to call rospy.spin()?

if __name__ == '__main__':
    main()
```

Saving that window: ***Control+O***
Exit from that window: ***Control+X***

13) Open a new terminal window  and run terminal window-1 by  command
    Command: ***python teleop.py***
14) We can see publishing that move command again

Some extra commands that I learned:

i) Odem topic provides what are the message type: package name.
   Command: ***rostopic info odem***

ii)

iii)

iv)

v)

vi)