**INTRODUCTION OF ROS**

**ROS Concepts:**

ROS has three levels of concepts: the Filesystem level, the Computation Graph level, and the Community level.

* + **ROS Filesystem level:**

Covers ROS resources

* + - **Packages:**  Packages are the main unit for organizing software in ROS. Packages are the most atomic build item and release item in ROS. Meaning that the most granular thing you can build and release is a package.
    - **Metapackages**: Metapackages are specialized Packages which only serve to represent a group of related other packages.
    - **Package Manifests**: Manifests (package.xml) provide metadata about a package, including its name, version, description, license information, dependencies, and other meta information like exported packages.
    - **Repositories:** A collection of packages which share a common VCS system (version control system).
    - **Message Types**: Message descriptions. Define the data structures for messages sent in ROS.
    - **Services (srv) types**: Service description. Define the request and response data structures for services in ROS.
  + **ROS Computational Graph level:**

The Computation Graph is the peer-to-peer network of ROS processes that are processing data together.

* + - **Nodes:** Nodes are the executables that communicate with each other.
    - **Master:**  provides name registration and lookup to the rest of the nodes in the ROS system. Without the Master, nodes would not be able to find each other, exchange messages, or invoke services.
    - **Parameter Server:** Allows data to be stored by key in a central location.
    - **Messages:** A message is simply a data structure comprising typed fields. Nodes communicate with each other by passing messages
    - **Topics:** Topics are named buses over which nodes  exchange messages.
    - **Service:** A ROS service is **a client/server system**. Here are some of the main characteristics of a ROS service: It is synchronous. The client sends a requests, and blocks until it receives a response. You should use ROS services only for computations and quick actions.
    - **Bags:** Bags are a format for saving and playing back ROS message data.
  + **ROS Community Level:**
    - **Distributions:** Collection of versioned stack that you can install**.**

**ROS Client Libraries:**

* Roscpp: c++ client library for ros.
* Rospy: pure python client library for ros.
* Roslisp: client library of LISP for ros.

**TUTORIALS**

* **Beginner Level:**

Way to check and ensure that environment variables like ROS\_ROOT(sets the location where the ROS core packages are installed) and ROS\_PACKAGE\_PATH (allows you to add more ROS packages from source to your environment).

**$ printenv | grep ROS**

* For Organizing and building your ROS code two methods are available : **rosbuild** and **catkin**

On UBUNTU when you install ROS you will have setup.\*sh files and now you need to source them with the command.

**$ source /opt/ros/noetic/setup.bash**

Where noetic is the short name of your ROS distribution other names are also available like kinetic , melodic etc.

**NOTE:** You will need to run this command on every new shell you open to have access to the ROS commands.

* **HOW TO CREATE A ROS WORKSPACE:**

**What is catkin?**

* catkin is **the official build system of ROS**
* The successor to the original ROS build system, rosbuild.
* catkin combines CMake macros and Python scripts to provide some functionality on top of CMake's normal workflow.

**Catkin is used for ROS Groovy and later. For ROS Fuerte and earlier ,select rosbuild.**

What is catkin workspace?

A catkin workspace is a folder where you modify, build and install catkin packages.

Let's create and build a catkin workspace:

$ mkdir -p ~/catkin\_ws/src

$ cd ~/catkin\_ws/

$ catkin\_make

**What is catkin\_make?**

The [catkin\_make](http://wiki.ros.org/catkin/commands/catkin_make) command is a convenience tool for working with [catkin workspaces](http://wiki.ros.org/catkin/workspaces).

Running it the first time in your workspace, it will create a CMakeLists.txt link in your 'src' folder (source folder).

It combines the calls to cmake and make in the standard Cmake workflow.

To make sure your workspace is properly overlayed by the setup script, make sure ROS\_PACKAGE\_PATH environment variable includes the directory you're in.

$ echo $ROS\_PACKAGE\_PATH

* **rospack**

rospack is a command-line tool for retrieving information about ROS packages available on the filesystem.

To find path to the package:

$ rospack find [package\_name]

* **roscd**

change directory starting with package, stack, or location name.

It is the part of rosbash.

Usage:

$ roscd <package-or-stack>[/subdir]

To print the working direcetory:

$ pwd

pwd(print working directory)

* **rosls**

list directly in a package by name rather than by absolute path.

$ rosls <package-or-stack>[/subdir]

* **Creating a ROS package**
* **Let’s first discuss about the Catkin package –**
  + **For a package to be considered a catkin package it must meet a few requirements:**
    - **It must contain a catkin compliant package.xml file –**
      * **package.xml file provides meta information about the package.**
    - **It must contain a CMakeLists.txt which uses catkin –**
      * **If it is a catkin metapackage it must have the relevant boilerplate CMakeLists.txt file.**
    - **Each package must have its own folder**
      * **No nested package nor multiple packages sharing the same directory**

**Structure of package:**

my\_package/ (package own folder)

CMakeLists.txt (text file)

package.xml (extensible markup language file)

* **Creating a catkin package –**
  + $ cd ~/catkin\_ws/src
  + $ catkin\_create\_pkg beginner\_tutorials std\_msgs rospy roscpp

**# This command will create a folder named beginner\_tutorials which contains a package.xml and CMakeLists.txt with partial filled information**

* **Building a catkin workspace –**

Now we will build our created workspace

* + $ cd ~/catkin\_ws
  + $ catkin\_make

**# This will build package in the catkin workspace**

* + $ . ~/catkin\_ws/devel/setup.bash

**# To add the workspace to your ROS environment you need to source the generated setup file**

* **Package Dependencies**
* **First-order Dependencies –**
  + **$ rospack depends1 beginner\_tutorials**

**# This will list all the first order dependencies that has been provided during the creation of catkin package (default dependencies).**

* + **$ roscd beginner\_tutorials**
  + **$ cat package.xml**

**# Dependencies for a package are stored in the package.xml file, the above command can fetch the package.xml file.**

* **Indirect Dependencies –**
  + **$ rospack depends1 rospy**

**# A dependency can also have its** own dependencies

* + **$ rospack depends beginner\_tutorials**

**# Lists all the nested dependencies**

* **Let’s Build your package –**
  + **$ cd ~/catkin\_ws/**

**# Move to the directory if you are not currently working in it.**

* + **$ ls src**

**# After this command we can see the beginner\_tutorials package we have created . We can now build that package using catkin\_make.**

* + **$ catkin\_make**

**# You will now be seeing lots of output from cmake and then make.**

* + **$ ls**

**# You will notice there are several folders have been created in your catkin\_workspace**

**i.e build, devel, src**

* **Build Folder –**
  + **It is the default location of the build space and is where cmake and make are called to configure and build your packages**
* **Devel Folder –**
  + **It is the default location of the devel space.**
  + **It is the location where your executables and libraries go before you install your packages.**
* **Understanding ROS Nodes**

**Description of graph concepts –**

* + **Nodes –** 
    - A node is an executable that uses ROS to communicate with other nodes.
    - ROS nodes use a ROS client library to communicate with other nodes.
      * **Client Libraries –** 
        + **rospy – python client library**
        + **roscpp – c++ client library**
    - Nodes can publish or subscribe to a Topic.
    - Nodes can also provide or use a Service.
  + **Messages –**
    - ROS datatype 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 (provides name service for ROS) + rosout (stdout/stderr) + parameter server (parameter server will be introduced later)
    - **roscore** is the first thing you should run when using ROS.

$ roscore

* **Publisher** –

ROS publisher is a ROS node that publishes a specific type of ROS message over a given ROS topic.

* **Subscriber-**

Subscribes to the Topic so that it receives the messages whenever any message is published to the Topic

* **Using rosnode**:

(nodes inside ros, use to get information about the node)

rosnode displays information about the ROS nodes that are currently running. The rosnode list command lists these active nodes:

$ rosnode list

The rosnode info command returns information about a specific node.

$ rosnode info /rosout

* **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).

$ rosrun turtlesim turtlesim\_node

turtlesim -package name

turtlesim\_node – node\_name

To rename node: $ rosrun turtlesim turtlesim\_node \_\_name:=my\_turtle

* Let's use another rosnode command, ping, to test that it's up:

**$ rosnode ping my\_turtle**

* **Turtle keyboard teleoperation**

For this first we need to run:

**$ roscore** then in new terminal we need to run

**$ rosrun turtlesim turtlesim\_node** then in new terminal we will run the command for turtle keyboard teleoperation i.e.

**$ rosrun turtlesim turtle\_teleop\_key**

* **ROS TOPICS:**
* **Using rqt\_graph**

A dynamic graph of what’s going on the system.

**$ rosrun rqt\_graph rqt\_graph**

* **Introduction of rostopic:**

The rostopic tool allows you to get information about ROS **topics**.

You can use the help option to get the available sub-commands for rostopic

$ rostopic -h

* rostopic bw display bandwidth used by topic
* rostopic echo print messages to screen
* rostopic hz display publishing rate of topic
* rostopic list print information about active topics
* rostopic pub publish data to topic
* rostopic type print topic type
* **Using rostopic echo:**

**$ rostopic echo /turtle1/cmd\_vel**

Where : /turtle1/cmd\_vel is the topic name

* **Using rostopic list:**

rostopic list returns a list of all topics currently subscribed to and published.

**$ rostopic list -h**

* **ROS Messages:**

Communication on topics happens by sending ROS **messages** between nodes. For the publisher (turtle\_teleop\_key) and subscriber (turtlesim\_node) to communicate, the publisher and subscriber must send and receive the same **type** of message.

* **Using rostopic type**

rostopic type returns the message type of any topic being published.

**$ rostopic type /turtle1/cmd\_vel**

**Our message type is geometry\_msgs/Twist**

We can look at the details of the message using rosmsg:

**$ rosmsg show geometry\_msgs/Twist**

* **Using rostopic pub**

rostopic pub publishes 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]'**

**$ rostopic pub /turtle1/cmd\_vel geometry\_msgs/Twist -r 1 -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, -1.8]'**

* **Using rostopic hz**

rostopic hz reports the rate at which data is published.

**$ rostopic hz /turtle1/pose**

* **Using rqt\_plot**

rqt\_plot displays a scrolling time plot of the data published on topics.

**$ rosrun rqt\_plot rqt\_plot**

* **ROS SERVICES :**

Services are another way that nodes can communicate with each other. Services allow nodes to send a **request** and receive a **response**.

**Using rosservice**

rosservice can easily attach to ROS's client/service framework with services.

rosservice list print information about active services

rosservice call call the service with the provided args

rosservice type print service type

rosservice find find services by service type

rosservice uri print service ROSRPC uri

YAML files: are usually used in ROS **to load Node configuration parameters in the ROS Parameter server**.