#### General Terms

* ***Nodes*** are executables, basically like scripts. Use rosrun to execute nodes
* ***Topics*** are name routes over which nodes communicate. Use rostopic to display information about topics.
* ***Messages*** are what nodes use to communicate with each other. Nodes publish messages to topics. It is composed of typed fields.
* ***Publishers*** and ***Subscribers*** are nodes that send messages into topics and to other nodes. Publishers send messages to topics and subscribers receive messages from topics. Subscribers usually invoke a callback function to process messages.
* ***Services*** are used when one node needs to inquire about a specific data. ex) robot needs to ask voice assistant about weather
  + ASK
* ***Parameter Server*** is a shared dictionary used by nodes to store and retrieve parameters at runtime.
  + Parameters can be 32-bit integers, booleans, strings, doubles, lists, etc
  + They are named using *ROS naming convention*
* ***ROS Master*** is responsible for communication between independent nodes. Maintains registries for nodes and helps them find each other.
* ***Bags*** are used to record, store, and playback messages from topics
* ROS Launch (see appendix 1)

##### Commands for general terms

\*see Foxy tutorial for more command line tools

* ***Roscore*** - used to start ROS master.
  + Should always be run first so that master is running as you use ROS
* ***Roslaunch*** - tool for launching multiple ROS nodes and setting parameters on Parameter Server.
  + - Can take in one or more XML configuration files (has .launch extension) that specifies parameter settings and which nodes to launch.
* ***Rospack*** - used to obtain information about packages
  + ex) rospack find rospy
* ***Roscd*** - change directories
  + ex) roscd <package>[/subdir]
* ***Rosls*** - ls directly in package without having to move into package
  + ex) rosls rospy → lists out different packages
* ***Rosrun*** - use to run executables from any package without having to move into it
  + ex) rosrun <package\_name> <executable\_file\_name>
* ***Rostopic*** - use to obtain info on topics
  + ex) rostopic <command>
  + <command> can be echo, info, list, or type
  + Echo prints messages to screen
  + Info prints info about active topic
  + List lists active topics
  + Type prints topic type
* ***Rosparam*** - parameter commands
  + ex) rosparam <command>
  + <command> can be list, get, set, or delete
  + Rosparam list → lists parameters’ names
  + Rosparam get <param-name> → get parameter’s value
  + Rosparam set <param-name> → set parameter to a value
  + Rosparam delete <param-name> → delete parameter
* ***Rosbag*** - used to record, playback messages from topics
  + Rosbag [TOPIC NAME]
  + Rosbag play [bagfile]
  + Rosbag play --clock [bagfile] – publish from a specific time in bagfile

#### ROS Environment Variables

* Parameters used in OS to specify operating environment of OS, such as location of system folder
* **ROS\_ROOT** : records location of ROS core packages
* **ROS\_MASTER\_URI** : helps nodes find master
* **ROS\_PACKAGE\_PATH** : records paths of packages
* **ROS\_IP** *or* **ROS\_HOSTNAME** : the IP address / hostname of a Node or another piece of equipment

#### 

#### ROS2 Foxy Tutorial

<https://docs.ros.org/en/foxy/Tutorials.html>

**Notes:**

* Downloaded binary package
* Set domain\_id as 0
* Set local\_host\_only to 1 (which does not let communication with other computers on network)

##### Beginner: CLI Tools

* *ros2 \_\_\_ list*
  + Use to list nodes, topics, services, etc
* *ros2 run <package\_name> <executable\_name>*
  + See Parameters for more info
  + Can add --ros-args for more arguments
    - This with --log-level <severity> displays specific logs for each severity level (see Rqt\_console Logs for more info)

###### Nodes

* *ros2 node info <node name>*
  + Gives info on that particle node
  + Returns a list of subscribers, publishers, services, and actions (the ROS graph connections)

###### Topics

* *ros2 topic list*
  + Lists topics
  + Adding -t to end will list parameters of topics and are useful to know what data the topics are about
* r*os2 topic echo <topic name>*
  + Use to display the information being conveyed through topic
  + <topic name> is name of topic or path if it is nested in something else
* *ros2 topic info <topic name>*
  + Displays info on this topic
* *ros2 interface show <msg type>*
  + Displays information on messages, specifically what data it should include from publisher for subscribers
  + These can be customized and documented
* *ros2 topic pub <topic\_name> <msg\_type> '<args>'*
  + Use in terminal to publish message to topic
  + <args> is the message and data you are sending to topic
  + ex) ros2 topic pub --once /turtle1/cmd\_vel geometry\_msgs/msg/Twist "{linear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.8}}"
    - “-- once” is just optional keyword specifying to do code once and exit
    - Can replace with other keywords to specify rate to send messages
  + All nodes are active and awaiting messages to do something, it’s not necessarily that the messages contain commands but data for the nodes to use and act upon.
* *ros2 topic hz /turtle1/pose*
  + See rate at which messages are being published

###### Services

* Services also work to relay data between nodes but unlike the continuous stream of a topic services only transmit data upon request
  + Can be only one service servers for a service, but many clients may use it
* *ros2 service list*
  + Use to see all service servers
* *ros2 service type <service\_name>*
  + Use to see the type of a service. Add -t to see all active services
  + ex) ros2 service type /clear
  + Type Empty means service sends no data upon request and receives no data with response.
* *ros2 service find <type\_name>*
  + Find all the services of a specific type
  + ex) find all services of type Empty by doing ros2 service find std\_srvs/srv/Empty
* *ros2 interface show <type\_name>.srv*
  + Display the structure of input arguments for a specific type
  + ex) ros2 interface show std\_srvs/srv/Empty.srv → - - -
    - The - - - means nothing since Empty is empty type
  + ros2 interface show turtlesim/srv/Spawn → bunch of args
* *ros2 service call <service\_name> <service\_type> <arguments>*
  + *<service\_name> -* the intended service name
  + *<service\_type> -* the type of said service
  + *<arguments> -* optional arguments

###### Parameters

* *ros2 param list*
  + Displays each node and their parameters
* *ros2 param get <node\_name> <parameter\_name>*
  + Display the type and current value of a parameter
  + ex) ros2 param get /turtlesim background\_g
* *ros2 param set <node\_name> <parameter\_name> <value>*
  + Change a parameter’s values at runtime
* *ros2 param dump <node\_name>*
  + Dump node’s parameter values into a file
  + Will save the file in current directory under *<node\_name>* **+** *.yaml*
* *ros2 param load <node\_name> <parameter\_file>*
  + Load parameters from a file into node
* *ros2 run <package\_name> <executable\_name> --ros-args --params-file <file\_name>*
  + Start a node while loading in parameters from file
  + Same as starting program but now you include extra flags
  + *<package\_name> -* package you are using
  + *<executable\_name> -* program you are running from package
  + *--ros-args -* needed to indicate new arguments being parsed
  + -*-params-file -* indicates param file is going to be passed through
  + *<file\_name> -* file you are parsing
* Got up to understanding actions

###### Actions

* Built on topics and servers, they function similarly to services except they can be canceled while executing.
* They also provide steady feedback
* Action client sends a goal to the action server which then sends back a stream of feedback and results.
  + The visualization of this looks very similar to services, the action ‘node’ includes two services, Goal and Result Service, that help facilitate the transfer of goals and feedback
* In the turtlesim tutorial, the letter key controls move the turtle to a specific orientation, these are the actions you can perform. But pressing the F key immediately halts the action, this is the unique feature of actions being cancelable.
  + In the /turtlesim node window you can see the feedback that is relayed as the turtle finishes a move or if a move is canceled
  + You can also preempt a move by performing two actions right after one another. Press on action key and then press another one, the first gets canceled and the second takes priority.
* Recall that *ros2 node info <node\_name>* will return all information regarding a node including the action servers and clients available
  + Of course, each node will be labeled as a server if there are things under “Action Servers”
  + If they are clients then there will be things under “Action Clients” that indicates
  + These things are what the node requests or gives results for
* *ros2 action list*
  + List all actions
  + Include -t to indicate the types of these actions (necessary when executing action from command line or code)
* *ros2 action info <action\_name>*
  + Get more information on a specific action
  + Will indicate the corresponding action servers and clients
* *ros2 interface show <action\_name>*
  + Display the data features of action
  + Split into two sections, with first being format of goal request and second the result
* *ros2 action send\_goal <action\_name> <action\_type> <values>*
  + Prompt an action
  + <values> - is the values you are sending with the request in YAML format
    - It’s an action so you gotta instruct action server to do something
    - The result is more just the server indicating that the request was completed and on the side it can give back some useful feedback.
  + Add --feedback to end to display the action server’s feedback
  + Each goal has a unique ID

###### Rqt\_console Logs

* Levels of severity in logger
  1. *Fatal -* messages indicate the system is going to terminate to try to protect itself from detriment.
  2. *Error* - messages indicate significant issues that won’t necessarily damage the system, but are preventing it from functioning properly.
  3. *Warn* - messages indicate unexpected activity or non-ideal results that might represent a deeper issue, but don’t harm functionality outright.
  4. *Info* - messages indicate event and status updates that serve as a visual verification that the system is running as expected.
  5. *Debug* - messages detail the entire step-by-step process of the system execution.
* These are not standard but a good gist

###### Launching Nodes

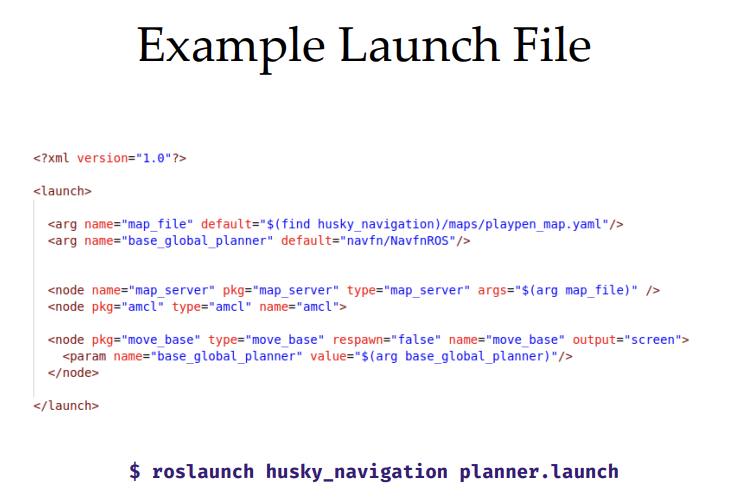
* Efficient to start up and configure a number of executables containing ROS2 nodes simultaneously.
* *ros2 launch <package\_name> <executable\_name>*
  + Run launch file that contains entire system of nodes and configurations
  + <package\_name> - name of package where file is located
  + <executable\_name> - name of file that contains nodes and configurations

###### Recording and Playing Back Data

* It is useful to record published data on topics for future reference and experiment recreation
* *ros2 bag* 
  + Command line tool to record this published data
* *ros2 bag record <topic\_name>*
  + First move into directory you wish to record data to, and then run this command
  + This will automatically store all data transmitted over topics to the directory
  + Files will be named with pattern *rosbag2\_year\_month\_day-hour\_minute\_second*
* *ros2 bag record -o <filename> <topic1\_name> <topic2\_name> . . .*
  + Can record multiple topics just by listing them out
  + The -o suffix is really only to provide a custom name to the file
  + Using -a will add all topics
* *ros2 bag info <bag\_file\_name>*
  + See info on recording, not the actual data but a description of the file and recording that took place
* *ros2 bag play <bag\_file\_name>*
  + Play the file and its data, make sure the no simulation is currently running

##### Beginner: Client Libraries

#### Appendix



1

