ROS commands:

1) Create a workspace directory and inside it a src directory, then initialise in the workspace directory with catkin_make. Source ROS's setup.bash and your workspace's setup.bash (from the devel folder).

2) Rospack - getting information about ROS packages
   rospack find <package_name>
   example - rospack find roscpp

3) Roscd - changing directory in ros
   example - roscd roscpp

4) Roscd, like other ROS tools, will only find directories listed within ROS_PACKAGE_PATH

5) Roscd log
   takes us to the folder where ros stores log files

6) Rosls
   listing ros files

7) What is a catkin package?
   a package.xml file for meta information
   CMakeLists.txt file which uses catkin
   each package must have its own folder

8) Catkin packages are in the src folder.

9) In the src folder, run the command
   catkin_create_pkg <package_name> <dependency 1> <dependency 2> … <dependency n>
   example -
   cd ~/catkin_ws/src
   catkin_create_pkg beginner_tutorials std_msgs rospy roscpp

10) Run catkin_make from the workspace folder to build the package that we just created.
    example -
    cd ~/catkin_ws
    catkin_make

11) Then source the setup file of the workspace again.

12) Finding dependencies of a package:
    rospack depends1 <package_name>
    We used depends1 for the first order dependencies. These dependencies are stored in the package.xml file of the created package.

13) Many dependencies have their own dependencies, but ROS can recursively determine all dependencies.
    rospack depends <package_name>

14) Customise your package.xml file:
    builtool_depend - catkin
    build_depend - the dependencies that we want at build time
    exec_depend - the dependencies that we want at run time

15) Build directory - where make and cmake are are called to configure and build the packages
devel folder - where executables and libraries go, before installing the packages

16) Nodes - an executable that ROS uses to communicate with other nodes
message - ros data type when publishing or subscribing to a topic
topic - nodes can subscribe to a topic or publish to it to send or receive messages
master - name service for ros (to help nodes find one another)
rosout - ros equivalent of stdout/stderr
Roscore - master + rosout + parameter server

17) Nodes can provide or use a service.

18) ROS client libraries allow nodes written in different programming languages to communicate.
rospy and roscpp

19) Roscore is the first thing we should run when using ros. (roscore = master naming service + rosout + parameter server)

20) Rosnode - used for information about the nodes that are currently running.

21) Rosnode list - shows the list of currently active nodes
example - rosnode list
Rosnode info - shows the info for a node
example - rosnode info /rosout
This command shows the topics that rosout is publishing to, subscribed to, and the services that rosout is related to.

22) Rosrun allows us to use the package name for running the node within that package
rosrun <package_name> <node_name>
example - rosrun turtlesim turtlesim_node

use the flag __name:=node_name to rename the new node to node_name
example - rosrun turtlesim turtlesim_node __name:=my_turtle

23) Use rosnode ping node_name to test if a node is up.

24) To control the turtlesim_node by keyboard, launch the node turtle_teleop_key
example - rosrun turtlesim turtle_teleop_key

25) In the turtlesim example, the turtle_teleop_key is a node which publishes the keystrokes to a topic and turtlesim_node subscribes to this topic.

26) Rqt_graph for a dynamic graph visualisation of what nodes and topics are currently running.
example - rosrun rqt_graph rqt_graph

27) Rostopic allows us to get information about ros topics
rostopic -h for help options

rostopic echo shows the data published on a topic
by echoing the data, we have created a node that is subscribed to the topic cmd_vel

rostopic list -v for a verbose list of the active ros topics, both the published and subscribed topics

28) The publisher and the subscriber communicate using the same type of message, that is, a topic has a type.

29) To find the type of a topic, use
rostopic type topic_name

30) To look at the details of a message:
rosmsg show message_name

31) To publish messages on a topic
rostopic pub topic_name msg_type arguments
example - rostopic pub -1 /turtle1/cmd_vel geometry_msg/Twist —‘[2.0, 0.0, 0.0]’ ‘[0.0, 0.0, 1.8]’
— tells that none of the following options is an option
-r flag to set rate (in Hz)
example - rostopic pub /turtle1/cmd_vel geometry_msg/Twist -r 1 —‘[2.0, 0.0, 0.0]’ ‘[0.0, 0.0, 1.8]’

32) Rostopic hz reports the rate at which data is being published
rostopic hz topic

33) Using rostopic type in conjugaltion with rosmsg show
example - rostopic type topic_name | rosmsg show

34) rqt_plot displays a scrolling time plot of the data being published to topics.

35) Rosrun rqt_plot rqt_plot

36) Services are another way for nodes to communicate. A node can send a request and receive a response.

37) Rosservice
rosservice list gives the list of available services

38) Rosservice type service_name
example -
rosservice type /clear returns std_srtvs/Empty which means that the service takes no arguments. (Sends no data while making a request and receives no data when receiving a response).
example -
rosservice type /spawn returns

float32 x
float32 y
float32 z
string name
———————
string name

so we can call it like : rosservice call /spawn 2 2 0.2 “” (since the name argument is optional)

39) Rosservice call service_name argument1 argument2 … argumentn
example - rosservice call /clear clears the background of the turtlesim_node.

40) Rosparam - allows us to store and manipulate data on the ROS parameter service. The parameter service can store integers, floats, booleans, dictionaries, lists.
rosparam uses the YAML markup language for syntax.

41) Rosparam get param_name
rosparam set param_name
example -

rosparam get /background_r returns 89
rosparam set /background_r 150 sets the value of background_r to 150.

42) This changes the parameter value, but we have to call the /clear service for the parameter change to take effect.

43) Rosparam get / returns the contents of the entire parameter server.
To store these values in a file, we can use -
rosparam dump file_name namespace
rosparam load file_name namespace

example -
rosparam dump params.yaml

We can even load these yaml files into new namespaces.
example -
rosparam load params.yaml copy
rosparam get /copy/background_r returns 255

44) rqt_console and rqt_logger_level
rqt_console attaches to ROS's logging framwork to display the output from nodes.
rqt_logger_level allows us to change the verbosity level of nodes as they run.

To run these:
rosrun rqt_console rqt_console
rosrun rqt_logger_level rqt_logger_level

45) To change the verbosity level from rqt_logger_level, refresh the nodes and then change the level.

46) Priority of logging levels:
Fatal (Highest priority)
Error
Warn
Info
Debug (lowest priority)

On selecting any level, messages of priority equal and higher are shown.
For example - on selecting warn, messages of priority warn, error and fatal will be shown.

47) Roslaunch starts nodes as defined in a launch file.
roslaunch package_name filename.launch

Create a 'launch' folder within the package folder and place the launchfile in there.

48) Editing the launchfile:

```
<launch>

  <group ns="turtlesim1">
    <node pkg="turtlesim" name="sim" type="turtlesim_node"/>
  </group>

  <group ns="turtlesim2">
    <node pkg="turtlesim" name="sim" type="turtlesim_node"/>
  </group>

  <node pkg="turtlesim" name="mimic" type="mimic">
    <remap from="input" to="turtlesim1/turtle1"/>
    <remap from="output" to="turtlesim2/turtle1"/>
  </node>

</launch>
```

49) The launch tags identify this file as a launchfile.
ns (namespace)
Two groups of two separate namespaces, turtlesim1 and turtlesim2 with a turtlesim node with a name of sim. This allows to start two simulators without having name conflicts.

50) We then start the mimic node with topic input and output renamed to turtlesim1 and turtlesim2. This renaming causes turtlesim2 to mimic turtlesim1.

51) Roslaunching the launchfile:
roslauch beginner_tutorials turtlemimic.launch

52) Nodes - ROS term for an executable that is connected to the ROS network.
A publisher is a talker node which continually broadcasts a message.

53) Writing a simple publisher in C++:
create a cpp file for the publisher in the src folder of the package.

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

http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29

What's happening in the chatter.cpp file:

- 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

54) Create a cpp file for the listener in the src folder of the package.
What's happening in the listener.cpp file:
- Initialize the ROS system
- Subscribe to the `chatter` topic
- Spin, waiting for messages to arrive
- When a message arrives, the `chatterCallback()` function is called

55) Recording data from ROS
Create a directory for the recorded data to be stored.
example - mkdir ~/bagfiles

56) All rosbag commands should be run from the bagfiles directory.

57) rosbag record -a (flag a stands for all messages from all nodes, all pubished messages should be recorded in a bag file)
Press Ctrl + C when we want to stop recording data

58) Rosbag info bag_file_name.bag
to show the information of the recorded .bag file.

59) To replay theh recorded information, use:
rosbag play bag_file_name.bag
The duration between running the play command and the actual play will be approximately equal to the time duration between the record command and the commands given.

To play at a different rate, use -r rate_here flag.
Example - rosbag play -r 2 bag_file_name.bag

60) Recording a subset of the data:
rosbag record -O file_name.bag /node_1 /node_2

This command tells rosbag to record the node_1 and node_2 data in a file called file_name.bag

61) Roscd then Roswtf is used for trying to find out problems.

62) Using the transform:
Transform means shifting of axes between different coordinate systems in a robot. This can either be done manually at each instance of handling the data (which can be a pain) or can be automatically handled by the tf library.

63)