Contenido

[Comandos de terminal de ROS 2](#_Toc118719221)

[Creación de los paquetes 2](#_Toc118719222)

[Ejecución de nodos 2](#_Toc118719223)

[Comando para paquetes 2](#_Toc118719224)

[Comandos para los topics 2](#_Toc118719225)

[Comandos para nodos 2](#_Toc118719226)

[Comando para mensajes personalizados 2](#_Toc118719227)

[Comando para servicios 2](#_Toc118719228)

[Comando para acciones 3](#_Toc118719229)

[Comandos de Rospy 3](#_Toc118719230)

[Nodos 3](#_Toc118719231)

[Topics 3](#_Toc118719232)

[Mensajes 3](#_Toc118719233)

[Bucles 3](#_Toc118719234)

[Servicios 4](#_Toc118719235)

[Servidor 4](#_Toc118719236)

[Clientes 4](#_Toc118719237)

[Acciones 4](#_Toc118719238)

[Servidor 4](#_Toc118719239)

[Cliente 4](#_Toc118719240)

[Topics 4](#_Toc118719241)

[Publicador 4](#_Toc118719242)

[Suscriptor 5](#_Toc118719243)

[Mensajes personalizados 5](#_Toc118719244)

[Servicios 6](#_Toc118719245)

[Cliente 6](#_Toc118719246)

[Servidor 6](#_Toc118719247)

[Mensajes personalizados 7](#_Toc118719248)

[Acciones 8](#_Toc118719249)

[Cliente 8](#_Toc118719250)

[Servidor 8](#_Toc118719251)

[Mensajes personalizados 10](#_Toc118719252)

[Comando de C++ 12](#_Toc118719253)

[Nodos 12](#_Toc118719254)

[Topics 12](#_Toc118719255)

[Mensajes 12](#_Toc118719256)

[Bucles 12](#_Toc118719257)

[Compilación de archivos 12](#_Toc118719258)

[Topics 12](#_Toc118719259)

[Publicador 12](#_Toc118719260)

[Suscriptor 13](#_Toc118719261)

[Mensajes personalizados 14](#_Toc118719262)

# Comandos de terminal de ROS

## Creación de los paquetes

catkin\_create\_package "nombre del paquete" rospyà Crea un paquete con rospy

catkin\_make à Compila todos los paquetes

catkin\_make --only\_pkg\_with\_deps “nombre\_del\_paquete”à Compila un paquete

## Ejecución de nodos

roscore à Inicia el nodo maestro de ROS

rosrun "nombre del paquete" "nombre del archivo" à ejecuta un programa de Ros

## Comando para paquetes

rospack list à Devuelve la lista de paquetes ROS

## Comandos para los topics

rostopic list àPara ver la lista de topics activos

rostopic echo "topic" à Para ver el contenido de un topic

rostopic info "topic" à Para ver la información de un topic

## Comandos para nodos

rosnode list à Devuelve la lista de nodos activos

rosnode info "nombre del nodo" à Devuelve la información de un nodo

## Comando para mensajes personalizados

rosmsg list à Devuelve la lista de todos los mensajes

rosmsg show "Nombre del mensaje" à Devuelve la información de un mensaje

rosmsg list | grep “Mensaje” à Devuelve la ruta de un mensaje

## Comando para servicios

rosservice list à Devuelve la lista de servicios activos

rosservice info "servicio" à Devuelve la información de un servicio

rossrv show "paquete/mensaje" à Devuelve la información de los argumentos de un servicio

## Comando para acciones

rostopic list à Devuelve la lista de topics y las acciones

\* "accion"/cancel à Cancela la acción

\* "accion"/feedback à Muestra el feedback

\* "accion"goal à Muestra el objetivo

\* "accion"/result à Muestra el resultado

\* "accion"/status à Muestra el estado de la acción

rostopic pub "nombre\_accion/accion" à Ejecuta la acción

rostopic info "acción" à Devuelve la información de la acción

rosmsg list | grep "acción" à Muestra la lista de mensajes de las acciones

\* "accion"/MensajeAction à

\* "accion"/MensajeActionFeedback à

\* "accion"/MensajeActionGoal à

\* "accion"/MensajeActionResult à

\* "accion"/MensajeFeedback à

\* "accion"/MensajeGoal à

\* "accion"/MensajeResult à

rosmsg show "accion"/Mensaje\_\_\_ “ à Para devolver información del mensaje

#### Python

# Comandos de Rospy

## Nodos

rospy.init\_node(" ") à Creación de un nodo

## Topics

“nombre” = Publisher("nombre", Tipo de mensaje, queue\_size=10) à Creación de un publicador

“nombre”.publish(mensaje) àPublica el mensaje en el topic

rospy.Subscriber("nombre", Tipo de mensaje, callback) à Creación de un suscriptor

## Mensajes

Tipos de mensajes básicos : Int32,Int64, String …

“nombre” = Int32()à Creación de un mensaje

“nombre”.”variable del mensaje” = “valor” à Asignación de un valor al mensaje

## Bucles

rospy.spin()àBucle infinito

“nombre”=rospy.Rate(“frecuencia”)à Crea una frecuencia para un bucle

rospy.is\_shutdown()à Se usa en un While not para finalizar el bucle con **Control+C**

rate.sleep()à Pausa para tener la frecuencia del bucle

## Servicios

### Servidor

“nombre”= rospy.Service(Nombre, Tipo de mensaje, callback )

### Clientes

rospy.wait\_for\_services("nombre") à Espera a que el servidor este activo

“nombre\_servicio”= rospy.ServiceProxy("/servicio", Tipo de mensaje) à Creación de un cliente para un servicio

“nombre” = “nombre\_servicio”(“petición”) à Envía la petición al servicio

## Acciones

### Servidor

“nombre”= actionlib.SimpleActionServer("nombre",”mensaje”,self.goal\_callback,False)à Crea el servidor

“nombre”.start() à Inicia el servidor

self.”nombre”.is\_preempt\_requested() à Comprueba si hay una cancelación anticipada

self.”nombre”.set\_preempted() à Establece el estado como cancelado anticipadamente

self.”nombre”.publish\_feedback(self.\_\_retroalimentacion) à Envia el feedback

self.”nombre”.set\_succeeded(self.\_\_resultado) à Envia el resultado

self.”nombre”.set\_aborted() à Establece el cestado como abortado

### Cliente

“nombre”= actionlib.SimpleActionClient (“nombre”,Tipo de mensaje) àCrea un cliente

“nombre”.wait\_for\_server() à Espera por el servidor de acción

“nombre”.send\_goal(“mensaje”,done\_cb=“callback\_resultado”,feedback\_cb=callback\_feedback)àenvia el mensaje

“nombre”.cancel\_goal() à Cancela el servidor anticipadamente

Status = “nombre”.get\_status() à Para obtener el estado de la acción con el siguiente código: 0 = pendiente , 1 = ejecutándose , 2= finalizado , 3= warning, 4 = error , 5 = cancelado

“nombre”.wait\_for\_result() à Espera al que el servidor termine de ejecutar la acción

# Topics

## Publicador

#!/usr/bin/env python

import rospy

from std\_msgs.msg import Int32

rospy.init\_node("Primer nodo") #Para crear un nodo

pub = rospy.Publisher("laser", Int32, queue\_size=10) #Crea el topic

a = Int32() #Comfiguramos a como un mensaje de tipo Int32

contador = 0 #Creamos una variable contador

rate = rospy.Rate(10) #Para crear un rate de 10Hz

while not rospy.is\_shutdown(): #Mientras no se cierre el nodo

    contador += 1

    a.data = contador

    pub.publish(a) #Publicamos el mensaje

    rate.sleep() #Esperamos para que sea la frecuencia de 10Hz

## Suscriptor

#!/usr/bin/env python

import rospy

from std\_msgs.msg import Int32

def callback(msg):

    """

    Funcion que se ejecuta cuando se recibe un mensaje

    """

    print ( msg )

rospy.init\_node('lectura')#Creamos el nodo

rospy.Subscriber('laser', Int32, callback) #Nos suscribimos al topic laser

rospy.spin() #Mantener el nodo activo

## Mensajes personalizados

1. Crear una carpeta msg en el paquete
2. Generamos un archivo “nombre”.msg y añadimos las variables del mensaje
3. Modificar el archivo CMakeLists.txt
4. En la función **find\_package ()** dentro de ella escribimos "**message\_generation**" y también añadimos los **paquetes** que use el mensaje creado (En líneas diferentes)
5. En la función **add\_message\_files ()** la descomentamos y añadimos el **nombre del archivo .msg** que creamos (En líneas diferentes)
6. En la función **generate\_messages ()** la descomentamos y **añadimos el nombre de los paquetes** que usamos en el mensaje creado
7. En la función **catkin\_package ()** la descomentamos y en la **línea de CATKIN\_DEPENDS añadimos rospy y message\_rutine**
8. Comprobamos si la función **include\_directories()** **esta descomentada y si ${catkin\_INCLUDE\_DIRS} esta dentro** de ella, pero dejamos comentado incluye

4.**Modificar package.xml**

1) En el final modificamos el archivo para que siga el siguiente esquema

**<build\_depend> rospy </build\_depend>**

**<build\_depend> message\_generation </build\_depend>**

**<build\_export\_depend> rospy </build\_export\_depend>**

**<build\_export\_depend> message\_rutine </build\_export\_depend>**

**<exec\_depend> rospy </exec\_depend>**

**<exec\_depend> message\_rutine </exec\_depend>**

5. **Compilar el paquete** ejecutando **catkin\_make --only\_pkg\_with\_deps “nombre\_del\_paquete”** dentro de la carpeta catkin\_ws

# Servicios

## Cliente

#!/usr/bin/env python

import rospy

from paquete.srv import Mensaje, MensajeRequest

rospy.init\_node('cliente\_servicio') #Crea el topic

rospy.wait\_for\_service('/nombre\_servicio')#Espera asta que el servicio se este ejecutando para ofrecer el servicio

#Se crea la conexion con el servicio

servicio = rospy.ServiceProxy("/servicio", "nombre del servicio")

#Se crea el objeto de "Tipo de mensaje" para enviarlo al servicio

peticion = MensajeRequest()

peticion.dato = "dato" #Asignacion de los datos a la peticion

#Se envia la peticion al servicio

resultado = servicio(peticion)

#Se imprime el resultado

print(resultado)

## Servidor

#!/usr/bin/env python

import rospy

from std\_srvs.srv import Empty, EmptyResponse #Importa el tipo de mensaje que necesitamos

def mi\_callback(request):

    """

    Funcion que sera ejecutada cuando se llame al servicio mi\_servicio

    """

    print("Peticion recibida")

    return (EmptyResponse) #Retorno un mensaje vacio

rospy.init\_node('servidor') #Inicializa el nodo

mi\_servicio = rospy.Service("mi\_servicio", Empty, mi\_callback) #Crea el servicio. Argumentos: Nombre del servicio, Tipo de mensaje, Funcion que se ejecuta cuando se recibe una peticion

rospy.spin() #Mantiene el nodo activo

## Mensajes personalizados

1. Crear una carpeta srv en el paquete del servidor del servicio
2. Creamos y editamos el fichero "Nombre".srv con el siguiente esquema

#Petición

int32 dato

---

#Respuesta

bool éxito

1. Modificamos el CMLists.txt para que se compile el servicio
2. En la función **find package()**, la descomentamos y **añadimos message\_generation** y añadimos las **dependencias** del archivo
3. En la función **add\_service\_files** **añadimos el nombre del archivo** de mensaje
4. En la función **generate\_messages**, la descomentamos y **añadimos las dependencias del archivo**
5. En la función **catkin\_package** **añadimos rospy y message\_runtime en la línea de CATKIN\_DEPENDS**

1. **Modificar package.xml**

1.En el final modificamos el archivo para que siga el siguiente esquema.

**<build\_depend> rospy </build\_depend>**

**<build\_depend> message\_generation </build\_depend>**

**<build\_export\_depend> rospy </build\_export\_depend>**

**<build\_export\_depend> message\_rutine </build\_export\_depend>**

**<exec\_depend> rospy </exec\_depend>**

**<exec\_depend> message\_rutine </exec\_depend>**

1. **Compilamos** el paquete usando **catkin\_make --only\_pkg\_with\_deps “nombre\_del\_paquete”** dentro de la carpeta catkin\_ws

# Acciones

## Cliente

#!/usr/bin/env python3

import rospy

import actionlib

from action\_server.msg import MensajeAction, MensajeGoal,MensajeResult , MensajeFeedback

def mi\_funcion(feedback):

    """

    Funcion que se ejecuta cuando el servidor envia feedback

    """

    print(feedback)

def acabado(result):

    """

    Funcion que se ejecuta cuando el servidor termina de ejecutar la accion

    """

    print("El resultado es: ", result.resultado)

#Inciamos el nodo de accion\_cliente

rospy.init\_node("accion\_cliente")

#Creamos el cliente de accion

cliente = actionlib.SimpleActionClient("accion\_servidor",MensajeAction)

#Esperamos al servidor de acciones

cliente.wait\_for\_server()

#Creamos el mensaje goal

goal = MensajeGoal()

goal.ciclos = 10

#Enviamos el mensaje goal

cliente.send\_goal(goal,done\_cb = acabado, feedback\_cb=mi\_funcion)

## Servidor

#!/usr/bin/env python3

import rospy

import actionlib

from servidor\_action.msg import MensajeAction,MensajeResult,MensajeFeedback #Importamos los mensajes de la accion desde el paquete donde estan definidos

class accion\_simple(object):

    #Creamos dos variables privadas que seran mensajes

    \_\_retroalimentacion = MensajeFeedback()

    \_\_resultado = MensajeResult()

    def \_\_init\_\_(self):

        #Creamos el servidor

        self.\_\_act\_serv= actionlib.SimpleActionServer("accion\_servidor",MensajeAction,self.goal\_callback,False)

        #Iniciamos el servidor

        self.\_\_act\_serv.start()

    def goal\_callback(self,datos):

        """Funcion que se ejecuta cuando se pide una accion al servidor

        RETURN: MensajeResult

        """

        #Creamos la variable con los datos que llegan a la accion

        Datos = datos.goal

        #Definimos la variable del estado de la accion

        succes = False

        #Definimos una variable par el estado de cancelado

        cancelado = False

        #Creamos un bucle

        while condicion :

            #<--> Acciones <-->

            #Comprobamos que no hay una cancelacion anticipateda

            if self.\_\_act\_serv.is\_preempt\_requested():

                #Fijamos el estado de la accion como cancelada anticipatedamente

                cancelado = True

                self.\_\_act\_serv.set\_preempted()

                break

            #Definimos el valor del feedback

            self.\_\_retroalimentacion.feedback = 1

            #Publicamos el valor del feedback

            self.\_\_act\_serv.publish\_feedback(self.\_\_retroalimentacion)

        #Comprobamos si el proceso esta finalizado

        if cancelado== False :

            if succes:

                #Definimos el valor del resultado

                self.\_\_resultado.resultado = 1

                #Ponemos el estado como finalizado y devolvemos el resultado

                self.\_\_act\_serv.set\_succeeded(self.\_\_resultado)

            else :

                #Definimos el estado como abortado pero si no fue cancelada anticipatedamente

                self.\_\_act\_serv.set\_aborted()

#FIXME:Modulo principal

rospy.init\_node("accion\_servidor")#Inciamos el nodo de accion\_servidor

accion\_simple() #Creamos la clase

rospy.spin()

## Mensajes personalizados

1. Creamos la carpeta action
2. Creamos el archivo “nombre”.action y añadimos los valores del mensaje

#Goal

---

#Resultado

---

#Feedback

1. Modificamos el CMakelists.txt
2. En la función **find\_package REQUIRED COMPONETS** añadimos **actionlib\_msgs** y **las librerias** de que depende el mensaje
3. En la función **add\_action\_files** añadimos los archivos "nombre".action
4. En la función **generate\_messages** añadimos **actionlib\_msgs** y las **librerías** que depende el mensaje
5. Comprobamos la función **catkin\_package** esta descomentada y **añadimos rospy**
6. Comprobamos que la función **include\_directories** esta descomentada
7. Modificaciones del package.xml
8. Añadimos las siguientes línea

**<build\_depend>actionlib\_msgs</build\_depend>**

**<build\_depend>actionlib\_msgs</build\_depend>**

**<build\_export\_depend>actionlib</build\_export\_depend>**

**<build\_export\_depend>actionlib\_msgs</build\_export\_depend>**

**<exec\_depend>actionlib</exec\_depend>**

**<exec\_depend>actionlib\_msgs</exec\_depend>**

1. **Compilamos** el paquete usando **catkin\_make --only\_pkg\_with\_deps “nombre\_del\_paquete”** dentro de la carpeta catkin\_ws

#### C++

# Comando de C++

### Nodos

ros::init(argc, argv ,”nombre”) 🡪 Crea el nodo

ros::NodeHandle nh;🡪 Crea el objeto NodeHandle

### Topics

ros:: Publisher “nombre” = nh.advertise<”Tipo de mensaje”>(“nombre”, “tamaño de la cola”); 🡪 Crea un publicador

ros::Subscriber “nombre” = nh.subscribe("topic", “tamaño cola”, “Callback”); 🡪 Crea un suscriptor

### Mensajes

“carpeta de mensaje”::”Tipo de mensaje” “nombre variable” 🡪 Crea un mensaje

“nombre”. “campo” 🡪 asigna un valor al campo

### Bucles

ros::spin() 🡪 Crea un bucle infinita

ros::Rate loop\_rate(“frecuencia”) 🡪 crea un bucle con frecuencia

ros::ok() 🡪 Comprueba en un bucle si se cancelo el proceso

# Compilación de archivos

Para compilar un archivo de roscpp tenemos que modificar el Cmakelist.txt en el apartado Build

Modificaciones:

1. Descomentamos la función add\_executable

**Resultado**: add\_executable(“nombre binario” src/”nombre archivo”.cpp)

1. Descomentamos la función add\_dependencies

**Resultado**: add\_dependencies(“nombre binario” ${${PROJECT\_NAME}\_EXPORTED\_TARGETS} ${catkin\_EXPORTED\_TARGETS})

1. Descomentamos la función target\_link\_libraries

**Resultado**: target\_link\_libraries(“nombre binario”

${catkin\_LIBRARIES}

)

1. Compilamos con el comando **catkin\_make --only\_pkg\_with\_deps “nombre\_del\_paquete”**
2. **Lanzamos** el binario con **rosrun “nombre paquete” “nombre binario”**

# Topics

## Publicador

#include <ros/ros.h>

#include <std\_msgs/Int32.h>

#include <iostream>

using namespace std;

int main(int argc, char \*\* argv){

    //Creamos el nodo principal del programa

    ros::init(argc, argv, "Nodo publicador");

    ros::NodeHandle nh;

    //Creamos el Publicador

    ros::Publisher pub = nh.advertise<std\_msgs::Int32>(contador, 1);

    //Creacion del mensaje

    std\_msgs::Int32 cont;

    cont.data = 0;

    //Creacion de la frecuencia de publicacion

    ros::Rate loop\_rate(1);

    //Bucle de publicacion

    while (ros::ok()){

        pub.publish(cont);

        cont.data++;

        loop\_rate.sleep();

    }

    return 0;

}

## Suscriptor

#include <ros/ros.h>

#include <std\_msgs/Int32.h>

void Callback(const std\_msgs::Int32::ConstPtr& msg)

{

    /\*Funcion que se ejecuta cuando llega un mensaje por el topic subcriptor\*/

    ROS\_INFO("I heard: [%d]", msg->data);

}

int mian(int argc, char \*\*argv){

    //Creamos el nodo principal

    ros::init(argc, argv, "subcriptor");

    ros::NodeHandle nh;

    //Creamos el subcriptor

    ros::Subscriber sub = nh.subscribe("topic", 1000, Callback);

    //Creacion del bucle infinito

    ros::spin();

    return 0;

}

## Mensajes personalizados

1. Creamos la carpeta msg
2. Creamos el archivo “nombre”.msg y añadimos los valores del mensaje
3. Modificamos el CMakelists.txt
4. En la función **find\_package REQUIRED COMPONETS** añadimos **message\_generation**
5. En la función **add\_message\_files** añadimos los archivos "nombre".msg
6. En la función **generate\_messages** añadimos las **librerías** que depende el mensaje
7. Comprobamos la función **catkin\_package** esta descomentada y **añadimos cpp y message\_runtime**
8. Modificaciones del package.xml
9. Añadimos las siguientes líneas al final

**<build\_depend>message\_generation</build\_depend>**

**<exec\_depend>message\_runtime</exec\_depend>**

1. **Compilamos** el paquete usando **catkin\_make --only\_pkg\_with\_deps “nombre\_del\_paquete”** dentro de la carpeta catkin\_ws

Para poder usarla en un paquete al generar el paquete es necesario añadirlo

catkin\_create\_pkg “nombre paquete ” roscpp std\_msgs “nombre del paquete”