Installation of ros2

# Colour:

Website

Software

Code

# Hardware Requirements (it is pre enabled in most of system) And Software

* **64-bit CPU** (x64 or ARM64).
* **Virtualization support** must be enabled in BIOS/UEFI:
  + Intel → **Intel VT-x** (Intel Virtualization Technology).
  + AMD → **SVM Mode** (Secure Virtual Machine).
  + Also enable **Intel VT-d / AMD-Vi** if available.

**1.2 GPU Drivers (if you want GUI / CUDA / ML / Gazebo)**

* For NVIDIA → Install the latest **NVIDIA Windows driver**   
  https://www.nvidia.com/en-in/drivers/
* For AMD → Install latest **Adrenalin drivers**.
* For Intel (iGPU/ARC) → Install the latest **Intel Graphics driver** (not Required).

**1.3 Net Frame work ( no need if already have)**

<https://support.microsoft.com/en-us/topic/microsoft-net-framework-4-8-offline-installer-for-windows-9d23f658-3b97-68ab-d013-aa3c3e7495e0>

**1.4 Python**

* + ROS 2 Humble/Iron recommended Python **3.10+** (64-bit).

https://www.python.org/downloads/

**1.5 Microsoft Visual c++**

https://learn.microsoft.com/en-us/cpp/windows/latest-supported-vc-redist?view=msvc-170

# Installation OF wsl for window

1.1 Window feature toun on off ; 1. **enable: Turn on window sub system linux ; 2.virtual machine platform 3.window hypervisor platform**

Press ok and restart system

**1.2 i. Search *window PowerShell* And Run As Administer**

**ii. type wsl –install Restart**

iii. wsl --list --online

iv. wsl –install ubuntu-22.04

v. set username Eg. dineshkolhedkk , pass- xyz

vi. Verify version and set wls version 2 wsl -l -v should see version 2

# Ros2 installation on windows wsl

We are going to install **ros2 Humble which support Linux 22.04 And It is Stable And tested**

1.1 <https://docs.ros.org/en/humble/Installation.html>  
 **or** (some time website Code, If You see it not same as shown)than I had past all code in ubuntu(source) copy past

# Ubuntu (source)

**Table of Contents**

System requirements

System setup

Set locale

Add the ROS 2 apt repository

Install development tools and ROS tools

Get ROS 2 code

Install dependencies using rosdep

Install additional DDS implementations (optional)

Build the code in the workspace

Environment setup

Source the setup script

Try some examples

Next steps after installing

Using the ROS 1 bridge

Additional RMW implementations (optional)

Alternate compilers

Clang

Stay up to date

Troubleshooting

Uninstall

## System requirements

The current Debian-based target platforms for Humble Hawksbill are:

Tier 1: Ubuntu Linux - Jammy (22.04) 64-bit

Tier 3: Ubuntu Linux - Focal (20.04) 64-bit Tier 3: Debian Linux - Bullseye (11) 64-bit

Other Linux platforms with varying support levels include:

Arch Linux, see [alternate instructions](https://wiki.archlinux.org/index.php/ROS#ROS_2)

Fedora Linux, see [alternate instructions](https://docs.ros.org/en/humble/Installation/Alternatives/Fedora-Development-Setup.html)

OpenEmbedded / webOS OSE, see [alternate instructions](https://github.com/ros/meta-ros/wiki/OpenEmbedded-Build-Instructions)

As defined in [REP 2000](https://www.ros.org/reps/rep-2000.html).

## System setup

### Set locale

Make sure you have a locale which supports UTF-8 . If you are in a minimal environment (such as a docker container), the locale may be something minimal like POSIX . We test with the following settings. However, it should be fine if you’re using a different UTF-8 supported locale.

|  |
| --- |
| $ locale # check for UTF-8  $ sudo apt update && sudo apt install locales  $ sudo locale-gen en\_US en\_US.UTF-8  $ sudo update-locale LC\_ALL=en\_US.UTF-8 LANG=en\_US.UTF-8  $ export LANG=en\_US.UTF-8  $ locale # verify settings |
|  |

### Add the ROS 2 apt repository

You will need to add the ROS 2 apt repository to your system.

First ensure that the [Ubuntu Universe repository](https://help.ubuntu.com/community/Repositories/Ubuntu) is enabled.

|  |
| --- |
| $ sudo apt install software-properties-common  $ sudo add-apt-repository universe |
|  |

The [ros-apt-source](https://github.com/ros-infrastructure/ros-apt-source/) packages provide keys and apt source configuration for the various ROS repositories.

Installing the ros2-apt-source package will configure ROS 2 repositories for your system. Updates to repository configuration will occur automatically when new versions of this package are released to the ROS repositories.

|  |
| --- |
| $ sudo apt update && sudo apt install curl -y  $ export ROS\_APT\_SOURCE\_VERSION=$(curl -s https://api.github.com/repos/rosinfrastructure/ros-apt-source/releases/latest | grep -F "tag\_name" | awk -F\" '{print $4}')  $ curl -L -o /tmp/ros2-apt-source.deb "https://github.com/ros-infrastructure/ros-aptsource/releases/download/${ROS\_APT\_SOURCE\_VERSION}/ros2-aptsource\_${ROS\_APT\_SOURCE\_VERSION}.$(. /etc/os-release && echo  $VERSION\_CODENAME)\_all.deb" # If using Ubuntu derivates use $UBUNTU\_CODENAME $ sudo dpkg -i /tmp/ros2-apt-source.deb |
|  |

### Install development tools and ROS tools

Install common packages.

|  |
| --- |
| $ sudo apt update && sudo apt install -y \ python3-flake8-docstrings \ python3-pip \ python3-pytest-cov \ ros-dev-tools |
|  |

Install packages according to your Ubuntu version.

**Ubuntu 22.04 LTS and later**

Ubuntu 20.04 LTS

$

sudo

apt

install

-

y

\

python3-flake8-blind-except

\

python3-flake8-builtins

\

python3-flake8-class-newline

\

python3-flake8-comprehensions

\

python3-flake8-deprecated

\

python3-flake8-import-order

\

python3-flake8-quotes

\

python3-pytest-repeat

\

python3-pytest-rerunfailures

## Get ROS 2 code

Create a workspace and clone all repos:

|  |
| --- |
| $ mkdir -p ~/ros2\_humble/src  $ cd ~/ros2\_humble  $ vcs import --input https://raw.githubusercontent.com/ros2/ros2/humble/ros2.repos src |
|  |

## Install dependencies using rosdep

ROS 2 packages are built on frequently updated Ubuntu systems. It is always recommended that you ensure your system is up to date before installing new packages.

|  |
| --- |
| $ sudo apt upgrade |
|  |
| $ sudo rosdep init  $ rosdep update  $ rosdep install --from-paths src --ignore-src -y --skip-keys "fastcdr rti-connextdds-6.0.1 urdfdom\_headers" |
|  |

**Note**: If you’re using a distribution that is based on Ubuntu (like Linux Mint) but does not identify itself as such, you’ll get an error message like Unsupported OS [mint] . In this case append --os=ubuntu:jammy to the above command.

## Install additional DDS implementations (optional)

If you would like to use another DDS or RTPS vendor besides the default, you can find instructions [here.](https://docs.ros.org/en/humble/Installation/RMW-Implementations.html)

## Build the code in the workspace

|  |
| --- |
| source |

|  |
| --- |
| /opt/ros/${ROS\_DISTRO}/setup.bash |

If you have already installed ROS 2 another way (either via debs or the binary distribution), make sure that you run the below commands in a fresh environment that does not have those other installations sourced. Also ensure that you do not have in your .bashrc . You can make sure that ROS 2 is not

sourced with the command printenv | grep -i ROS . The output should be empty.

More info on working with a ROS workspace can be found in [this tutorial](https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Colcon-Tutorial.html).

|  |
| --- |
| $ cd ~/ros2\_humble/  $ colcon build --symlink-install |
|  |

Note: if you are having trouble compiling all examples and this is preventing you from completing a successful build, you can use COLCON\_IGNORE in the same manner as [CATKIN\_IGNORE](https://github.com/ros-infrastructure/rep/blob/master/rep-0128.rst) to ignore the subtree or remove the folder from the workspace. Take for instance: you would like to avoid installing the large OpenCV library. Well then simply run touch COLCON\_IGNORE in the cam2image demo directory to leave it out of the build process.

# Environment setup(NO Need To Do This Follow next Step)

## Source the setup script

Set up your environment by sourcing the following file.

|  |
| --- |
| $ . ~/ros2\_humble/install/local\_setup.bash |
|  |

**Note**

Replace .bash with your shell if you’re not using bash. Possible values are: setup.bash , setup.sh , setup.zsh .

## Try some examples

In one terminal, source the setup file and then run a C++ talker :

|  |
| --- |
| $ . ~/ros2\_humble/install/local\_setup.bash  $ ros2 run demo\_nodes\_cpp talker |
|  |

In another terminal source the setup file and then run a Python listener :

|  |
| --- |
| $ . ~/ros2\_humble/install/local\_setup.bash  $ ros2 run demo\_nodes\_py listener |
|  |

|  |
| --- |
| I |

|  |
| --- |
| heard |

You should see the talker saying that it’s Publishing messages and the listener saying those messages. This verifies both the C++ and Python APIs are working properly.

Hooray!

## Next steps after installing

Continue with the [tutorials and demos](https://docs.ros.org/en/humble/Tutorials.html) to configure your environment, create your own workspace and packages, and learn ROS 2 core concepts.

## Using the ROS 1 bridge

The ROS 1 bridge can connect topics from ROS 1 to ROS 2 and vice-versa. See the dedicated [documentation](https://github.com/ros2/ros1_bridge/blob/master/README.md) on how to build and use the ROS 1 bridge.

## Additional RMW implementations (optional)

The default middleware that ROS 2 uses is Fast DDS , but the middleware (RMW) can be replaced at runtime. See the [guide](https://docs.ros.org/en/humble/How-To-Guides/Working-with-multiple-RMW-implementations.html) on how to work with multiple RMWs.

## Alternate compilers

Using a different compiler besides gcc to compile ROS 2 is easy. If you set the environment variables CC and CXX to executables for a working C and C++ compiler, respectively, and retrigger CMake configuration (by using --cmake-force-configure or by deleting the packages you want to be affected), CMake will reconfigure and use the different compiler.

### Clang

To configure CMake to detect and use Clang:

|  |
| --- |
| $ sudo apt install clang  $ export CC=clang $ export CXX=clang++  $ colcon build --cmake-force-configure |
|  |

**Stay up to date**

See [Maintain source checkout](https://docs.ros.org/en/humble/Installation/Maintaining-a-Source-Checkout.html) to periodically refresh your source installation.

**Troubleshooting**

Troubleshooting techniques can be found [here.](https://docs.ros.org/en/humble/How-To-Guides/Installation-Troubleshooting.html#linux-troubleshooting)

## Uninstall

1. If you installed your workspace with colcon as instructed above, “uninstalling” could be just a matter of opening a new terminal and not sourcing the workspace’s setup file. This way, your environment will behave as though there is no Humble install on your system.
2. If you’re also trying to free up space, you can delete the entire workspace directory with:

|  |
| --- |
| $ rm -rf ~/ros2\_humble |
|  |

**Step 1**

**1.1 sudo apt install ros-humble-desktop**

**1.2 source /opt/ros/humble/setup.bash**

**1.3 ros2**

**1.4 check which operator working**

echo $ROS\_DISTRO

**#output humble**

This runs a script (setup.bash) that sets up your terminal to recognize ROS 2 commands(Loads a script into your terminal).

|  |
| --- |
|  |

**Step2 optional**

Copy code

echo "source /opt/ros/humble/setup.bash" >> ~/.bashrc

You can make ROS 2 available **automatically** in every terminal by adding this line to your ~/.bashrc file: This means:

* **Write this line** (source /opt/ros/humble/setup.bash) **into** the file ~/.bashrc.
* The >> operator means: **append (add at the end)** of the file.

This way, every time you open a terminal, it will automatically set up ROS 2.

**Step2.2**

**Then activate it:**

**source ~/.bashrc**

Now every time you open a terminal, the ROS 2 environment is ready!

This command runs (executes) the .bashrc file immediately in the current terminal.Why? Because if you just edited .bashrc, you want to apply the changes right now, without **restarting the terminal.**

| **Question** | **Answer** |
| --- | --- |
| Do you need to set up the environment? | ✅ Yes, every terminal session |
| How do you do it? | source /opt/ros/humble/setup.bash |
| How to make it automatic? | Add to ~/.bashrc |
| What if you skip it? | ROS 2 commands won’t work |

Step 3**: Running Your First ROS 2 Program (Talker & Listener)**

**at You Will Learn:**

You’ll run two built-in ROS 2 programs:

* One **publishes messages** (called a “talker”)
* The other **receives messages** (called a “listener”)

They communicate using a **topic** called /chatter.

**🛠️ A. Open Terminal 1 and run the talker node:**

bash

Copy code

**ros2 run demo\_nodes\_cpp talker**

📤 This node will **publish messages** like:

[INFO] Publishing: 'Hello World: 1'

[INFO] Publishing: 'Hello World: 2'

**🛠️ B. Open Terminal 2 and run the listener node:**

Copy code

**ros2 run demo\_nodes\_cpp listener**

📥 This node will **receive the messages** and show:

[INFO] I heard: 'Hello World: 1'

[INFO] I heard: 'Hello World: 2'

**✅ What's happening?**

| **Role** | **Node** | **Action** | **Topic used** |
| --- | --- | --- | --- |
| Talker | Publisher | Sends out messages | /chatter |
| Listener | Subscriber | Listens to those messages | /chatter |

Both nodes **do not know each other directly**, but they talk using the topic system.

**What's happening?**

| **Role** | **Node** | **Action** | **Topic used** |
| --- | --- | --- | --- |
| Talker | Publisher | Sends out messages | /chatter |
| Listener | Subscriber | Listens to those messages | /chatter |

Both nodes **do not know each other directly**, but they talk using the topic system.

**📋 Want to check what's going on?**

While both nodes are running, in a **third terminal**, try:

**🔹 List active topics:**

Copy

**ros2 topic list**

You’ll see:

/chatter

/parameter\_events

/rosout

**🔹 See messages on /chatter:**

CopyEdit

**ros2 topic echo /chatter**

You’ll see:

data: Hello World: 5

This proves that your ROS 2 nodes are working and communicating! ✅

**📌 Summary: Your First ROS 2 Run**

| **Step** | **Command** | **Purpose** |
| --- | --- | --- |
| 1 | ros2 run demo\_nodes\_cpp talker | Run publisher node |
| 2 | ros2 run demo\_nodes\_cpp listener | Run subscriber node |
| 3 | ros2 topic list | View active topics |
| 4 | ros2 topic echo /chatter | Watch message traffic |

**echo /chatter**

You're probably confusing two different things:

**🚫 Wrong:**

bash

CopyEdit

echo /chatter

This command will just print the text /chatter to the screen:

bash

CopyEdit

/chatter

It **does NOT** interact with ROS 2.

**✅ Correct ROS 2 command:**

bash

CopyEdit

ros2 topic echo /chatter

**🔍 What this does:**

This tells ROS 2:

“Show me the messages being sent on the topic /chatter in real time.”

So if you have the **talker node** running, you will see output like:

data: Hello World: 1

data: Hello World: 2

...

This command is extremely useful to **debug or observe communication between nodes**.

**✅ Summary**

| **Command** | **What it does** |
| --- | --- |
| echo /chatter | Just prints the text "/chatter" (Linux shell only) |
| ros2 topic echo /chatter | Displays real-time messages on the /chatter topic (ROS 2 command) |

**Step 4 : Run the Turtlesim Robot (GUI-based ROS 2 Simulator)**

**📌 What is Turtlesim?**

**Turtlesim** is a simple 2D turtle robot simulator. It helps you learn ROS 2 topics, messages, and commands in a visual way.

**✅ A. Start the Turtle GUI**

In a new terminal:

CopyEdit

**ros2 run turtlesim turtlesim\_node**

You will see a window with a turtle on it.

This is a **ROS 2 node** that subscribes to motion commands (like move forward, turn, etc.).

**✅ B. Control the Turtle with Keyboard**

Open another terminal:

CopyEdit

**ros2 run turtlesim turtle\_teleop\_key**

Now **use the arrow keys** to move the turtle!

* ⬆️ Up arrow = move forward
* ⬇️ Down arrow = move backward
* ⬅️ Left arrow = rotate left
* ➡️ Right arrow = rotate right

**💡 What's Happening?**

|  |  |  |
| --- | --- | --- |
| Node | Role | Topic Used |
| turtlesim\_node | Robot simulator | /turtle1/cmd\_vel |
| turtle\_teleop\_key | Keyboard controller | Sends velocity commands |

So just like the talker/listener, these two nodes **talk to each other using a topic**, but this time the turtle **actually moves!**

**What is ROS?**

ROS, or Robot Operating System, is a set of open-source software libraries and tools that enables you to develop robot applications. Let’s explore its key aspects and importance in the field of robotics.

**Governance and Ecosystem**

NVIDIA joined the Open Source Robotics Alliance (OSRA) as a founding member and platinum sponsor. OSRA is an initiative by Open Source Robotics Foundation to foster collaboration, innovation and technical guidance in the robotics community by supporting several open-source robotics projects, including the Robot Operating System (ROS).

**Architecture and Components**

ROS functions as a middleware distributed systems framework. Its architecture consists of:

* **ROS packages:** Units of work that contain one or more nodes
* **Nodes:** Individual components that can be assembled into graphs
* **Graphs:** Represent the overall application structure

Nodes in a ROS system handle various tasks, such as:

* Interfacing with cameras
* Managing executive behavior
* Performing motion planning
* Communicating with actuators

These nodes can be interconnected in flexible ways, allowing for modular and reusable robotics software development.

**Ecosystem and Tools**

The ROS ecosystem includes a wide range of tools and packages, some of which have been made open-source by their developers and while others have built packages that are proprietary. Some notable examples include:

* [**MoveIt**](https://picknik.ai/moveit/)**:** A trajectory planning framework developed by Picknik
* [**Nav2**](https://docs.nav2.org/)**:** A navigation stack for Autonomous Mobile Robots (AMRs)

ROS 2 continues to evolve, with ongoing efforts to improve its capabilities for production robotics and integration with hardware acceleration technologies like those offered by NVIDIA - and this is a great reason for roboticists to embrace ROS.

As a roboticist, you probably don’t want to be thinking too much about your middleware. By leveraging ROS, roboticists can focus on developing advanced capabilities rather than reinventing basic middleware functionality. This standardization also promotes collaboration and knowledge sharing within the robotics community.

Webpage: https://learn.nvidia.com/courses/course?course\_id=course-v1:DLI+S-OV-34+V1&unit=block-v1:DLI+S-OV-34+V1+type@vertical+block@d52d7ce3cb3c4c248da2d72ce2042f74

Website: https://nvidia-isaac-ros.github.io/getting\_started/index.html