Tooling in ⁝⁝⁝ 2

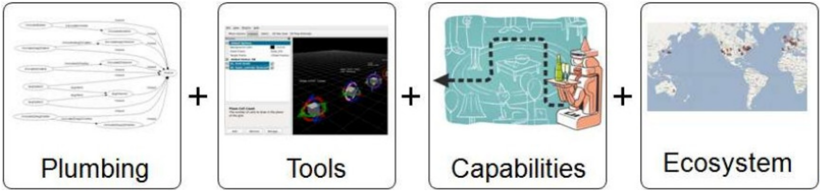
Olivier Kermorgant

ANF ROS2

CENTRALE NANTES

LS2N

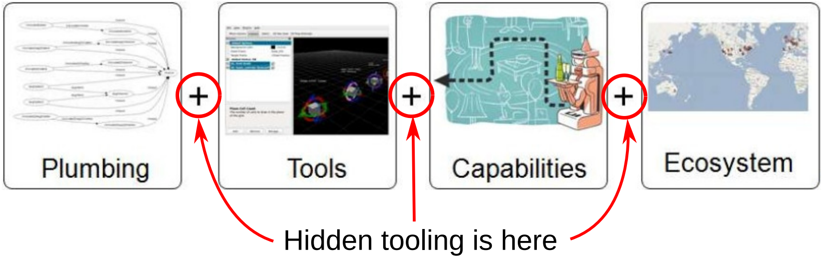
What kind of tooling do we need?



File system structure
Environment variables
(super)-build tools
Network behavior
Packaging

Workspaces
magic of `setup.bash`
colcon
Tuning DDS
bloom

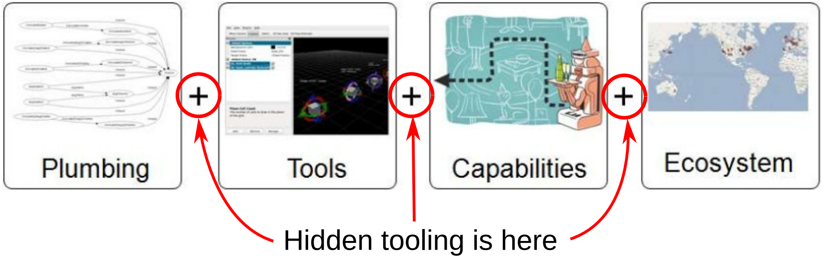
What kind of tooling do we need?



File system structure
Environment variables
(super)-build tools
Network behavior
Packaging

Workspaces
magic of `setup.bash`
colcon
Tuning DDS
bloom

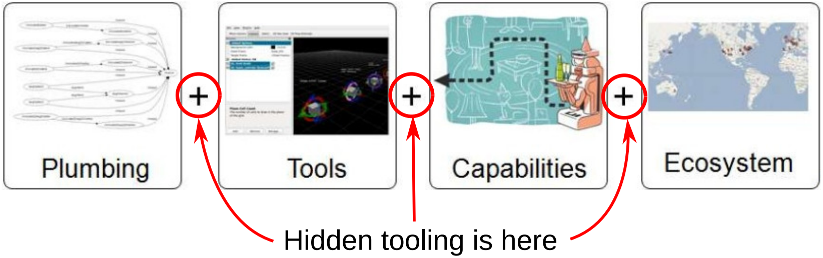
What kind of tooling do we need?



File system structure
Environment variables
(super)-build tools
Network behavior
Packaging

Workspaces
magic of `setup.bash`
colcon
Tuning DDS
bloom

What kind of tooling do we need?



File system structure

Environment variables

(super)-build tools

Network behavior

Packaging

Workspaces

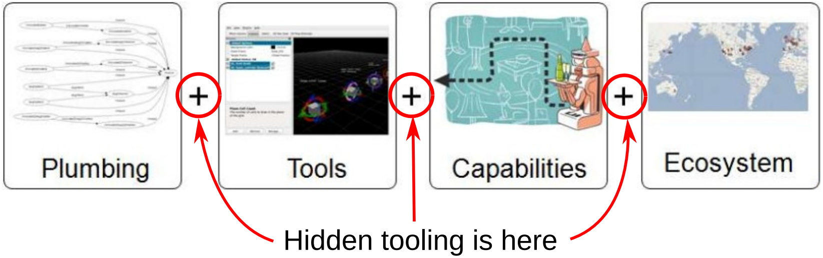
magic of `setup.bash`

colcon

Tuning DDS

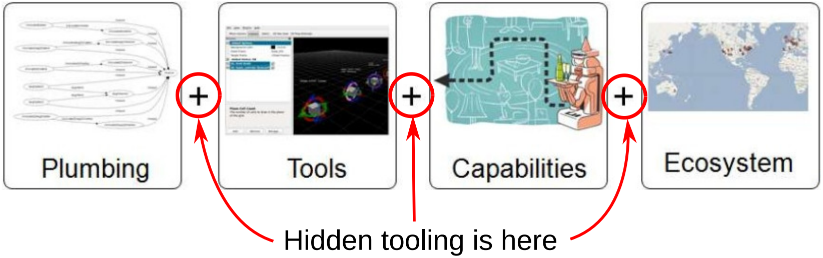
bloom

What kind of tooling do we need?



File system structure	Workspaces
Environment variables	magic of `setup.bash`
(super)-build tools	colcon
Network behavior	Tuning DDS
Packaging	bloom

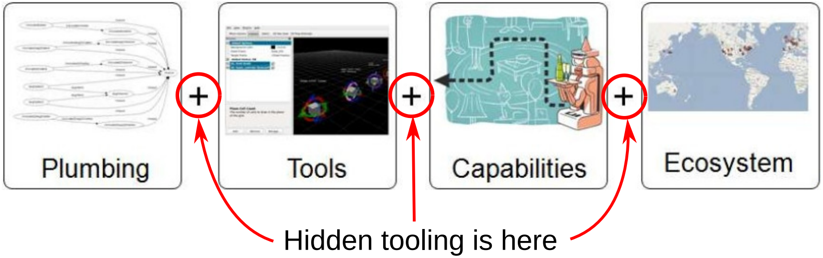
What kind of tooling do we need?



File system structure
Environment variables
(super)-build tools
Network behavior
Packaging

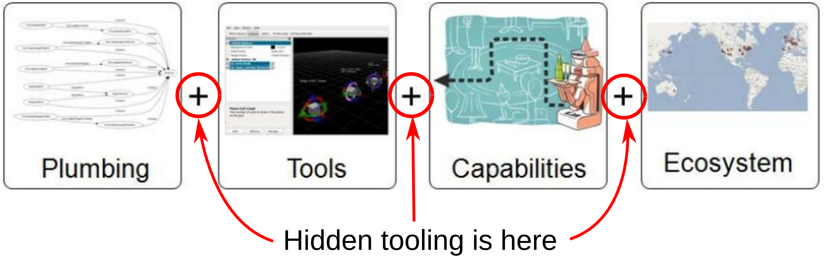
Workspaces
magic of `setup.bash`
colcon
Tuning DDS
bloom

What kind of tooling do we need?



File system structure	Workspaces
Environment variables	magic of `setup.bash`
(super)-build tools	colcon
Network behavior	Tuning DDS
Packaging	bloom

What kind of tooling do we need?



File system structure

Environment variables

(super)-build tools

Network behavior

Packaging

Workspaces

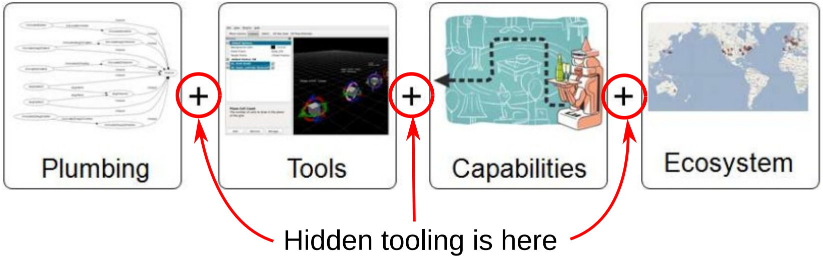
magic of `setup.bash`

colcon

Tuning DDS

bloom

What kind of tooling do we need?



File system structure

Environment variables

(super)-build tools

Network behavior

Packaging

Workspaces

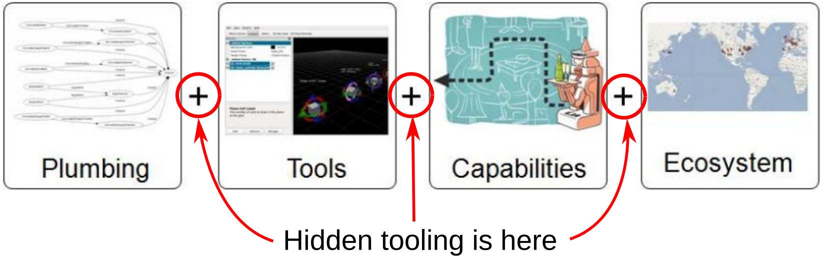
magic of `setup.bash`

colcon

Tuning DDS

bloom

What kind of tooling do we need?



File system structure

Environment variables

(super)-build tools

Network behavior

Packaging

Workspaces

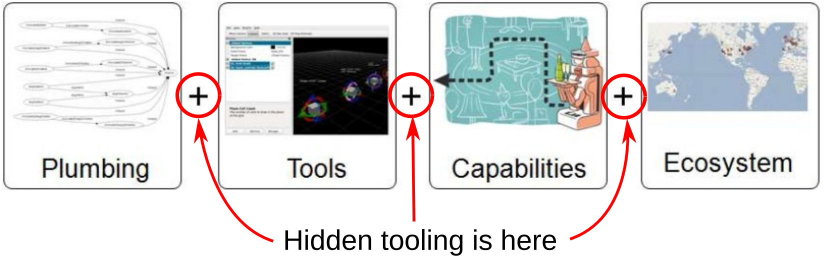
magic of `setup.bash`

colcon

Tuning DDS

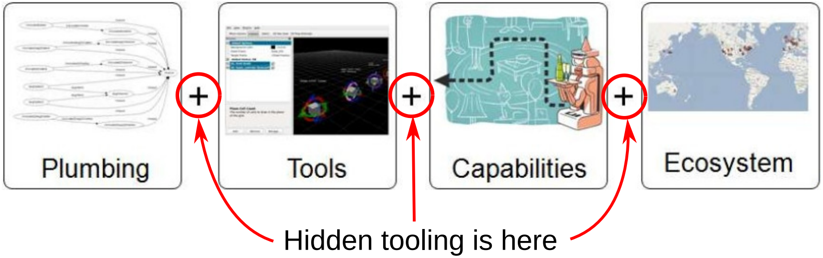
bloom

What kind of tooling do we need?



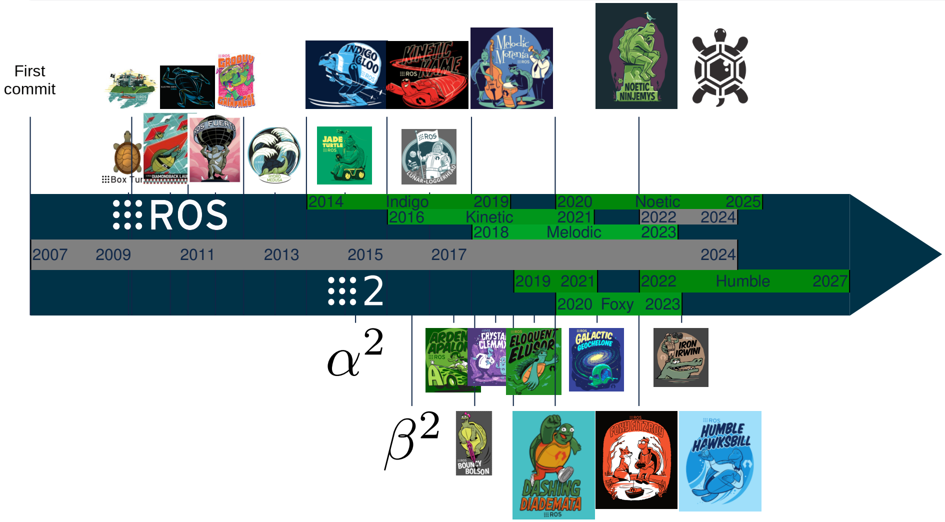
File system structure	Workspaces
Environment variables	magic of `setup.bash`
(super)-build tools	colcon
Network behavior	Tuning DDS
Packaging	bloom

What kind of tooling do we need?



File system structure	Workspaces
Environment variables	magic of `setup.bash`
(super)-build tools	colcon
Network behavior	Tuning DDS
Packaging	bloom

History of distributions - Long Term Support are what you want



In 2017: 200000 commits made by more than 2800 users

More than 2000 forks of `rosdistro` from package developpers

Any ROS file is part of a given package

- Atomic way to share and identify code
- Can be CMake-based or pure Python

A package is identified by its `package.xml` file

- Give the name + dependencies (other ROS packages or other libraries)

```xml
<?xml version="1.0"?>
<package format="3">
  <name>simulation_2d</name>
  <version>2.0.0</version>
  <description>The simulation2D package</description>
  <maintainer email="olivier.kermorgant@ec-nantes.fr">Olivier Kermorgant</maintainer>

  <license>MIT</license>
  <buildtool_depend>ament_cmake</buildtool_depend>

  <depend>geometry_msgs</depend>
  <depend>rclcpp</depend>
  <depend>sensor_msgs</depend>
  <depend>urdfdom</depend>

  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>
```

Any ROS file is part of a given package

- Atomic way to share and identify code
- Can be CMake-based or pure Python

A package is identified by its `package.xml` file

- Give the name + dependencies (other ROS packages or other libraries)

```xml
<?xml version="1.0"?>
<package format="3">
  <name>simulation_2d</name>
  <version>2.0.0</version>
  <description>The simulation2D package</description>
  <maintainer email="olivier.kermorgant@ec-nantes.fr">Olivier Kermorgant</maintainer>

  <license>MIT</license>
  <buildtool_depend>ament_cmake</buildtool_depend>

  <depend>geometry_msgs</depend>
  <depend>rclcpp</depend>
  <depend>sensor_msgs</depend>
  <depend>urdfdom</depend>

  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>
```

A package may contain:

- C++ code → `include/ src/`
- Python code→ `scripts/`
- robot descriptions → `urdf/ meshes/`
- launch files → `launch/`
- custom messages → `msg/ srv/`
- actually any file

C++ packages: `CMakeLists.txt`

Pure Python: `setup.cfg`

Running a node requires its package name

- `ros2 run package node`

File system - ROS packages - what's inside?

A package may contain:

- C++ code → `include/` `src/`
- Python code→ `scripts/`
- robot descriptions → `urdf/` `meshes/`
- launch files → `launch/`
- custom messages → `msg/` `srv/`
- actually any file

C++ packages: `CMakeLists.txt`

Pure Python: `setup.cfg`

Running a node requires its package name

- `ros2 run package node`

File system - ROS packages - what's inside?

A package may contain:

- C++ code $\rightarrow$ `include/` `src/`
- Python code $\rightarrow$ `scripts/`
- robot descriptions $\rightarrow$ `urdf/` `meshes/`
- launch files $\rightarrow$ `launch/`
- custom messages $\rightarrow$ `msg/` `srv/`
- actually any file

C++ packages: `CMakeLists.txt`

Pure Python: `setup.cfg`

Running a node requires its package name

- `ros2 run package node`

File system - ROS packages - what's inside?

A package may contain:

- C++ code → `include/` `src/`
- Python code→ `scripts/`
- robot descriptions → `urdf/` `meshes/`
- launch files → `launch/`
- custom messages → `msg/` `srv/`
- actually any file

C++ packages: `CMakeLists.txt`

Pure Python: `setup.cfg`

Running a node requires its package name

- `ros2 run package node`

File system - ROS packages - what's inside?

A package may contain:

- C++ code → `include/` `src/`
- Python code→ `scripts/`
- robot descriptions → `urdf/` `meshes/`
- launch files → `launch/`
- custom messages → `msg/` `srv/`
- actually any file

C++ packages: `CMakeLists.txt`

Pure Python: `setup.cfg`

Running a node requires its package name

- `ros2 run package node`

File system - ROS packages - what's inside?

A package may contain:

- C++ code $\rightarrow$ `include/` `src/`
- Python code$\rightarrow$ `scripts/`
- robot descriptions $\rightarrow$ `urdf/` `meshes/`
- launch files $\rightarrow$ `launch/`
- custom messages $\rightarrow$ `msg/` `srv/`
- actually any file

C++ packages: `CMakeLists.txt`

Pure Python: `setup.cfg`

Running a node requires its package name

- `ros2 run package node`

File system - ROS packages - what's inside?

A package may contain:

- C++ code → `include/  src/`
- Python code→ `scripts/`
- robot descriptions → `urdf/  meshes/`
- launch files → `launch/`
- custom messages → `msg/  srv/`
- actually any file

C++ packages: `CMakeLists.txt`

Pure Python: `setup.cfg`

Running a node requires its package name

- `ros2 run package node`

File system - ROS packages - what's inside?

A package may contain:

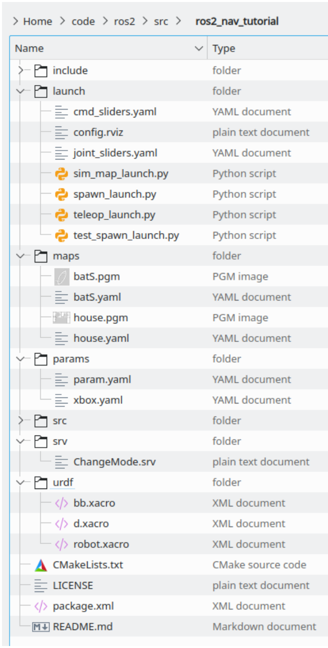
- C++ code → `include/` `src/`
- Python code→ `scripts/`
- robot descriptions → `urdf/` `meshes/`
- launch files → `launch/`
- custom messages → `msg/` `srv/`
- actually any file

C++ packages: `CMakeLists.txt`

Pure Python: `setup.cfg`

Running a node requires its package name

- `ros2 run package node`



> Home > code > ros2 > src > **ros2_nav_tutorial**

Name	Type
include	folder
launch	folder
cmd_sliders.yaml	YAML document
config.rviz	plain text document
joint_sliders.yaml	YAML document
sim_map_launch.py	Python script
spawn_launch.py	Python script
teleop_launch.py	Python script
test_spawn_launch.py	Python script
maps	folder
batS.pgm	PGM image
batS.yaml	YAML document
house.pgm	PGM image
house.yaml	YAML document
params	folder
param.yaml	YAML document
xbox.yaml	YAML document
src	folder
srv	folder
ChangeMode.srv	plain text document
urdf	folder
bb.xacro	XML document
d.xacro	XML document
robot.xacro	XML document
CMakeLists.txt	CMake source code
LICENSE	plain text document
package.xml	XML document
README.md	Markdown document

File system - ROS packages - what's inside?

A package may contain:

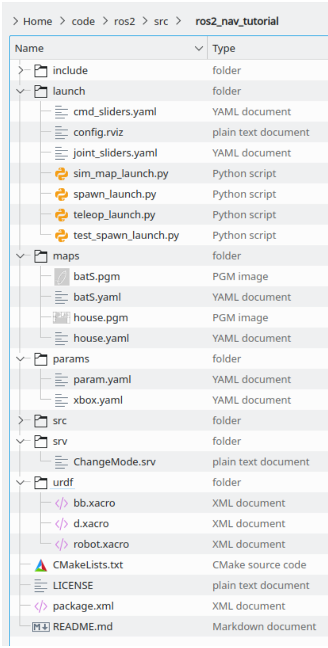
- C++ code → `include/` `src/`
- Python code→ `scripts/`
- robot descriptions → `urdf/` `meshes/`
- launch files → `launch/`
- custom messages → `msg/` `srv/`
- actually any file

C++ packages: `CMakeLists.txt`

Pure Python: `setup.cfg`

Running a node requires its package name

- `ros2 run package node`



> Home > code > ros2 > src > **ros2_nav_tutorial**

Name	Type
include	folder
launch	folder
cmd_sliders.yaml	YAML document
config.rviz	plain text document
joint_sliders.yaml	YAML document
sim_map_launch.py	Python script
spawn_launch.py	Python script
teleop_launch.py	Python script
test_spawn_launch.py	Python script
maps	folder
batS.pgm	PGM image
batS.yaml	YAML document
house.pgm	PGM image
house.yaml	YAML document
params	folder
param.yaml	YAML document
xbox.yaml	YAML document
src	folder
srv	folder
ChangeMode.srv	plain text document
urdf	folder
bb.xacro	XML document
d.xacro	XML document
robot.xacro	XML document
CMakeLists.txt	CMake source code
LICENSE	plain text document
package.xml	XML document
README.md	Markdown document

File system - ROS packages - what's inside?

A package may contain:

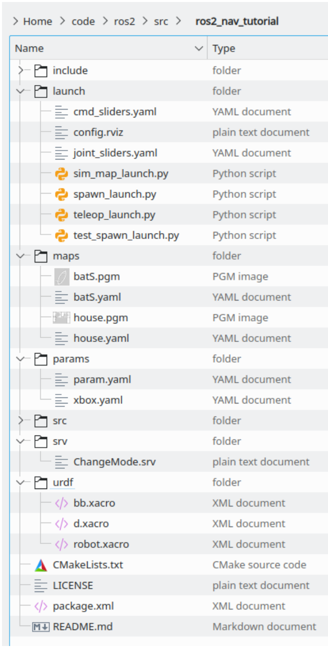
- C++ code → `include/` `src/`
- Python code→ `scripts/`
- robot descriptions → `urdf/` `meshes/`
- launch files → `launch/`
- custom messages → `msg/` `srv/`
- actually any file

C++ packages: `CMakeLists.txt`

Pure Python: `setup.cfg`

Running a node requires its package name

- `ros2 run package node`



> Home > code > ros2 > src > **ros2_nav_tutorial**

Name	Type
include	folder
launch	folder
cmd_sliders.yaml	YAML document
config.rviz	plain text document
joint_sliders.yaml	YAML document
sim_map_launch.py	Python script
spawn_launch.py	Python script
teleop_launch.py	Python script
test_spawn_launch.py	Python script
maps	folder
batS.pgm	PGM image
batS.yaml	YAML document
house.pgm	PGM image
house.yaml	YAML document
params	folder
param.yaml	YAML document
xbox.yaml	YAML document
src	folder
srv	folder
ChangeMode.srv	plain text document
urdf	folder
bb.xacro	XML document
d.xacro	XML document
robot.xacro	XML document
CMakeLists.txt	CMake source code
LICENSE	plain text document
package.xml	XML document
README.md	Markdown document

File system - ROS packages - what's inside?

A package may contain:

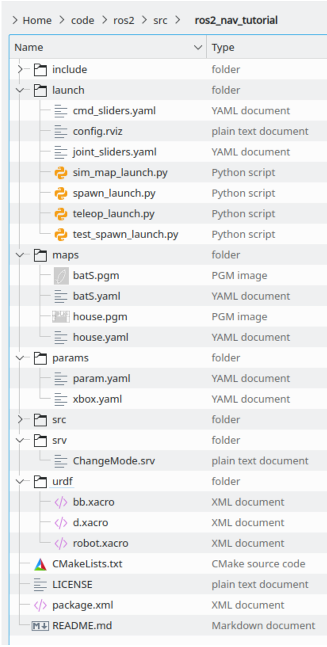
- C++ code → `include/` `src/`
- Python code→ `scripts/`
- robot descriptions → `urdf/` `meshes/`
- launch files → `launch/`
- custom messages → `msg/` `srv/`
- actually any file

C++ packages: `CMakeLists.txt`

Pure Python: `setup.cfg`

Running a node requires its package name

- `ros2 run package node`



> Home > code > ros2 > src > ros2_nav_tutorial

Name	Type
include	folder
launch	folder
cmd_sliders.yaml	YAML document
config.rviz	plain text document
joint_sliders.yaml	YAML document
sim_map_launch.py	Python script
spawn_launch.py	Python script
teleop_launch.py	Python script
test_spawn_launch.py	Python script
maps	folder
batS.pgm	PGM image
batS.yaml	YAML document
house.pgm	PGM image
house.yaml	YAML document
params	folder
param.yaml	YAML document
xbox.yaml	YAML document
src	folder
srv	folder
ChangeMode.srv	plain text document
urdf	folder
bb.xacro	XML document
d.xacro	XML document
robot.xacro	XML document
CMakeLists.txt	CMake source code
LICENSE	plain text document
package.xml	XML document
README.md	Markdown document

File system - ROS packages - what's inside?

A package may contain:

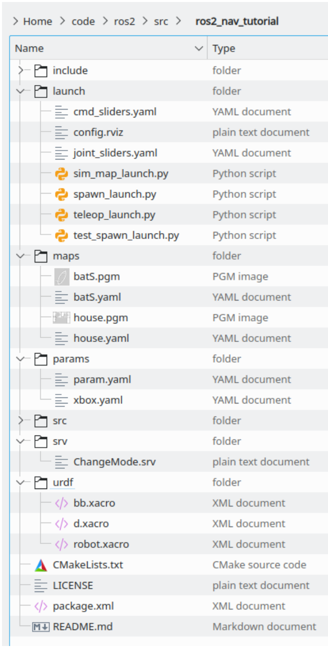
- C++ code $\rightarrow$ `include/` `src/`
- Python code $\rightarrow$ `scripts/`
- robot descriptions $\rightarrow$ `urdf/` `meshes/`
- launch files $\rightarrow$ `launch/`
- custom messages $\rightarrow$ `msg/` `srv/`
- actually any file

C++ packages: `CMakeLists.txt`

Pure Python: `setup.cfg`

Running a node requires its package name

- `ros2 run package node`



> Home > code > ros2 > src > **ros2_nav_tutorial**

Name	Type
include	folder
launch	folder
cmd_sliders.yaml	YAML document
config.rviz	plain text document
joint_sliders.yaml	YAML document
sim_map_launch.py	Python script
spawn_launch.py	Python script
teleop_launch.py	Python script
test_spawn_launch.py	Python script
maps	folder
batS.pgm	PGM image
batS.yaml	YAML document
house.pgm	PGM image
house.yaml	YAML document
params	folder
param.yaml	YAML document
xbox.yaml	YAML document
src	folder
srv	folder
ChangeMode.srv	plain text document
urdf	folder
bb.xacro	XML document
d.xacro	XML document
robot.xacro	XML document
CMakeLists.txt	CMake source code
LICENSE	plain text document
package.xml	XML document
README.md	Markdown document

File system - ROS packages - what's outside?

Packages have to be placed in specific directories: *workspaces*

- The system should know where to find packages
- A whole workspace can be compiled in a single command
- Classical sub-folders: `build install src`

A given terminal only knows about *sourced* workspaces

```
source /opt/humble/setup.bash
source /some/other/workspace/install/setup.bash
source ~/my_main_ws/install/setup.bash
```

- Can be done in `~/.bashrc`
- Careful when using GUI applications

Notion of overlay: sourcing order opposed to lookup order

- Packages and overlays are cheap
- Last sourced workspace overrides any existing package
- Useful to test a new feature even on official packages

File system - ROS packages - what's outside?

Packages have to be placed in specific directories: *workspaces*

- The system should know where to find packages
- A whole workspace can be compiled in a single command
- Classical sub-folders: `build install src`

A given terminal only knows about *sourced* workspaces

```
source /opt/humble/setup.bash
source /some/other/workspace/install/setup.bash
source ~/my_main_ws/install/setup.bash
```

- Can be done in `~/.bashrc`
- Careful when using GUI applications

Notion of overlay: sourcing order opposed to lookup order

- Packages and overlays are cheap
- Last sourced workspace overrides any existing package
- Useful to test a new feature even on official packages

File system - ROS packages - what's outside?

Packages have to be placed in specific directories: *workspaces*

- The system should know where to find packages
- A whole workspace can be compiled in a single command
- Classical sub-folders: `build install src`

A given terminal only knows about *sourced* workspaces

```
source /opt/humble/setup.bash
source /some/other/workspace/install/setup.bash
source ~/my_main_ws/install/setup.bash
```

- Can be done in `~/.bashrc`
- Careful when using GUI applications

Notion of overlay: sourcing order opposed to lookup order

- Packages and overlays are cheap
- Last sourced workspace overrides any existing package
- Useful to test a new feature even on official packages

Packages have to be placed in specific directories: *workspaces*

- The system should know where to find packages
- A whole workspace can be compiled in a single command
- Classical sub-folders: `build install src`

A given terminal only knows about *sourced* workspaces

```
source /opt/humble/setup.bash
source /some/other/workspace/install/setup.bash
source ~/my_main_ws/install/setup.bash
```

- Can be done in `~/.bashrc`
- Careful when using GUI applications

Notion of overlay: sourcing order opposed to lookup order

- Packages and overlays are cheap
- Last sourced workspace overrides any existing package
- Useful to test a new feature even on official packages

Packages have to be placed in specific directories: *workspaces*

- The system should know where to find packages
- A whole workspace can be compiled in a single command
- Classical sub-folders: `build install src`

A given terminal only knows about *sourced* workspaces

```
source /opt/humble/setup.bash
source /some/other/workspace/install/setup.bash
source ~/my_main_ws/install/setup.bash
```

- Can be done in `~/.bashrc`
- Careful when using GUI applications

Notion of overlay: sourcing order opposed to lookup order

- Packages and overlays are cheap
- Last sourced workspace overrides any existing package
- Useful to test a new feature even on official packages

Packages have to be placed in specific directories: *workspaces*

- The system should know where to find packages
- A whole workspace can be compiled in a single command
- Classical sub-folders: `build install src`

A given terminal only knows about *sourced* workspaces

```
source /opt/humble/setup.bash
source /some/other/workspace/install/setup.bash
source ~/my_main_ws/install/setup.bash
```

- Can be done in `~/.bashrc`
- Careful when using GUI applications

Notion of overlay: sourcing order opposed to lookup order

- Packages and overlays are cheap
- Last sourced workspace overrides any existing package
- Useful to test a new feature even on official packages

Packages have to be placed in specific directories: *workspaces*

- The system should know where to find packages
- A whole workspace can be compiled in a single command
- Classical sub-folders: `build install src`

A given terminal only knows about *sourced* workspaces

```
source /opt/humble/setup.bash
source /some/other/workspace/install/setup.bash
source ~/my_main_ws/install/setup.bash
```

- Can be done in `~/.bashrc`
- Careful when using GUI applications

Notion of overlay: sourcing order opposed to lookup order

- Packages and overlays are cheap
- Last sourced workspace overrides any existing package
- Useful to test a new feature even on official packages

File system - ROS packages - what's outside?

Packages have to be placed in specific directories: *workspaces*

- The system should know where to find packages
- A whole workspace can be compiled in a single command
- Classical sub-folders: `build install src`

A given terminal only knows about *sourced* workspaces

```
source /opt/humble/setup.bash
source /some/other/workspace/install/setup.bash
source ~/my_main_ws/install/setup.bash
```

- Can be done in `~/.bashrc`
- Careful when using GUI applications

Notion of overlay: sourcing order opposed to lookup order

- Packages and overlays are cheap
- Last sourced workspace overrides any existing package
- Useful to test a new feature even on official packages

File system - ROS packages - what's outside?

Packages have to be placed in specific directories: *workspaces*

- The system should know where to find packages
- A whole workspace can be compiled in a single command
- Classical sub-folders: `build` `install` `src`

A given terminal only knows about *sourced* workspaces

```
source /opt/humble/setup.bash
source /some/other/workspace/install/setup.bash
source ~/my_main_ws/install/setup.bash
```

- Can be done in `~/.bashrc`
- Careful when using GUI applications

Notion of overlay: sourcing order opposed to lookup order

- Packages and overlays are cheap
- Last sourced workspace overrides any existing package
- Useful to test a new feature even on official packages

Packages have to be placed in specific directories: *workspaces*

- The system should know where to find packages
- A whole workspace can be compiled in a single command
- Classical sub-folders: `build install src`

A given terminal only knows about *sourced* workspaces

```
source /opt/humble/setup.bash
source /some/other/workspace/install/setup.bash
source ~/my_main_ws/install/setup.bash
```

- Can be done in `~/.bashrc`
- Careful when using GUI applications

Notion of overlay: sourcing order opposed to lookup order

- Packages and overlays are cheap
- Last sourced workspace overrides any existing package
- Useful to test a new feature even on official packages

Packages have to be placed in specific directories: *workspaces*

- The system should know where to find packages
- A whole workspace can be compiled in a single command
- Classical sub-folders: `build install src`

A given terminal only knows about *sourced* workspaces

```
source /opt/humble/setup.bash
source /some/other/workspace/install/setup.bash
source ~/my_main_ws/install/setup.bash
```

- Can be done in `~/.bashrc`
- Careful when using GUI applications

Notion of overlay: sourcing order opposed to lookup order

- Packages and overlays are cheap
- Last sourced workspace overrides any existing package
- Useful to test a new feature even on official packages

Sourcing a workspace appends to numerous environment variables

```
CMAKE_PREFIX_PATH
PYTHONPATH
LD_LIBRARY_PATH
PATH
...
```

Mixing ROS 1 & 2: sourcing both ROS 1 and ROS 2 workspaces

- Numerous cryptic compilation or runtime errors

Some tools helps dealing with the two

```
# define ROS 1 and ROS 2 workspaces
ros1_workspaces="/opt/ros/noetic ~/code/libs/ros ~/code/ros"
ros2_workspaces="/opt/ros/foxy ~/code/libs/ros2 ~/code/ros2"
# source the tool
source ros_management.bash
# activate ROS 2 after cleaning environment variables from ROS 1
ros2ws
```

Sourcing a workspace appends to numerous environment variables

```
CMAKE_PREFIX_PATH
PYTHONPATH
LD_LIBRARY_PATH
PATH
...
```

Mixing ROS 1 & 2: sourcing both ROS 1 and ROS 2 workspaces

- Numerous cryptic compilation or runtime errors

Some tools helps dealing with the two

```
# define ROS 1 and ROS 2 workspaces
ros1_workspaces="/opt/ros/noetic ~/code/libs/ros ~/code/ros"
ros2_workspaces="/opt/ros/foxy ~/code/libs/ros2 ~/code/ros2"
# source the tool
source ros_management.bash
# activate ROS 2 after cleaning environment variables from ROS 1
ros2ws
```

Sourcing a workspace appends to numerous environment variables

```
CMAKE_PREFIX_PATH
PYTHONPATH
LD_LIBRARY_PATH
PATH
...
```

Mixing ROS 1 & 2: sourcing both ROS 1 and ROS 2 workspaces

- Numerous cryptic compilation or runtime errors

Some tools helps dealing with the two

```
# define ROS 1 and ROS 2 workspaces
ros1_workspaces="/opt/ros/noetic ~/code/libs/ros ~/code/ros"
ros2_workspaces="/opt/ros/foxy ~/code/libs/ros2 ~/code/ros2"
# source the tool
source ros_management.bash
# activate ROS 2 after cleaning environment variables from ROS 1
ros2ws
```

Sourcing a workspace appends to numerous environment variables

```
CMAKE_PREFIX_PATH
PYTHONPATH
LD_LIBRARY_PATH
PATH
...
```

Mixing ROS 1 & 2: sourcing both ROS 1 and ROS 2 workspaces

- Numerous cryptic compilation or runtime errors

Some tools helps dealing with the two

```bash
# define ROS 1 and ROS 2 workspaces
ros1_workspaces="/opt/ros/noetic ~/code/libs/ros ~/code/ros"
ros2_workspaces="/opt/ros/foxy ~/code/libs/ros2 ~/code/ros2"
# source the tool
source ros_management.bash
# activate ROS 2 after cleaning environment variables from ROS 1
ros2ws
```

Network behavior - defaults

ROS 1 needs a ROS master / roscore

- Not running by default
- Multi-computer possible if same master URI

Defaults in ROS 2 are the opposite

- No master
- Auto-discovery
- On the whole network

Network behavior - defaults

ROS 1 needs a ROS master / roscore

- Not running by default
- Multi-computer possible if same master URI

Defaults in ROS 2 are the opposite

- No master
- Auto-discovery
- On the whole network

ROS 1 needs a ROS master / roscore

- Not running by default
- Multi-computer possible if same master URI

Defaults in ROS 2 are the opposite

- No master
- Auto-discovery
- On the whole network

Network behavior - defaults

ROS 1 needs a ROS master / roscore

- Not running by default
- Multi-computer possible if same master URI

Defaults in ROS 2 are the opposite

- No master
- Auto-discovery
- On the whole network

Network behavior - defaults

ROS 1 needs a ROS master / roscore

- Not running by default
- Multi-computer possible if same master URI

Defaults in ROS 2 are the opposite

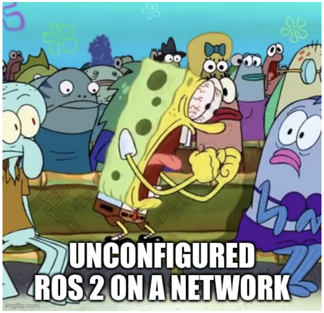
- No master
- Auto-discovery
- On the whole network

ROS 1 needs a ROS master / roscore

- Not running by default
- Multi-computer possible if same master URI

Defaults in ROS 2 are the opposite

- No master
- Auto-discovery
- On the whole network



Limiting traffic to localhost

```
export ROS_LOCALHOST_ONLY=1
```

Connect only with some other computers

```
unset ROS_LOCALHOST_ONLY # we want the network
export ROS_DOMAIN_ID=42 # only those will see me
```

Autodiscovery is still here

- Can be disabled (depends on DDS vendor...)

Limiting traffic to localhost

```
export ROS_LOCALHOST_ONLY=1
```

Connect only with some other computers

```
unset ROS_LOCALHOST_ONLY # we want the network
export ROS_DOMAIN_ID=42 # only those will see me
```

Autodiscovery is still here

- Can be disabled (depends on DDS vendor...)

Limiting traffic to localhost

```
export ROS_LOCALHOST_ONLY=1
```

Connect only with some other computers

```
unset ROS_LOCALHOST_ONLY # we want the network
export ROS_DOMAIN_ID=42 # only those will see me
```

Autodiscovery is still here

- Can be disabled (depends on DDS vendor...)

Limiting traffic to localhost

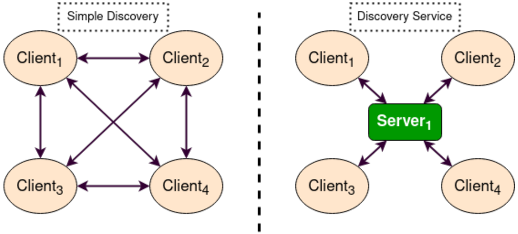
```
export ROS_LOCALHOST_ONLY=1
```

Connect only with some other computers

```
unset ROS_LOCALHOST_ONLY # we want the network
export ROS_DOMAIN_ID=42 # only those will see me
```

Autodiscovery is still here

- Can be disabled (depends on DDS vendor...)



Network behavior - more tuning

Limiting traffic to any network interface

- Depends on DDS vendor: editing a few XML

```
ros_restrict ETH # with ros_management_tools
```

Fine-tuning discovery: `ROS_AUTOMATIC_DISCOVERY_RANGE`

- since Iron

```
SUBNET # default: any reachable node
LOCALHOST # deprecates ROS_LOCALHOST_ONLY
OFF # no auto-discovery
```

- Discovery is not connection!
  - A node may wait to be discovered by another one

Limiting traffic to any network interface

- Depends on DDS vendor: editing a few XML

```
ros_restrict ETH # with ros_management_tools
```

Fine-tuning discovery: `ROS_AUTOMATIC_DISCOVERY_RANGE`

- since Iron

```
SUBNET # default: any reachable node
LOCALHOST # deprecates ROS_LOCALHOST_ONLY
OFF # no auto-discovery
```

- Discovery is not connection!
  - A node may wait to be discovered by another one

Network behavior - more tuning

Limiting traffic to any network interface

- Depends on DDS vendor: editing a few XML

```
ros_restrict ETH # with ros_management_tools
```

Fine-tuning discovery: `ROS_AUTOMATIC_DISCOVERY_RANGE`

- since Iron

```
SUBNET # default: any reachable node
LOCALHOST # deprecates ROS_LOCALHOST_ONLY
OFF # no auto-discovery
```

- Discovery is not connection!
  - A node may wait to be discovered by another one

Limiting traffic to any network interface

- Depends on DDS vendor: editing a few XML

```
ros_restrict ETH # with ros_management_tools
```

Fine-tuning discovery: `ROS_AUTOMATIC_DISCOVERY_RANGE`

- since Iron

```
SUBNET # default: any reachable node
LOCALHOST # deprecates ROS_LOCALHOST_ONLY
OFF # no auto-discovery
```

- Discovery is not connection!
    - A node may wait to be discovered by another one

Limiting traffic to any network interface

- Depends on DDS vendor: editing a few XML

```
ros_restrict ETH # with ros_management_tools
```

Fine-tuning discovery: `ROS_AUTOMATIC_DISCOVERY_RANGE`

- since Iron

```
SUBNET # default: any reachable node
LOCALHOST # deprecates ROS_LOCALHOST_ONLY
OFF # no auto-discovery
```

- Discovery is not connection!
  - A node may wait to be discovered by another one

Limiting traffic to any network interface

- Depends on DDS vendor: editing a few XML

```
ros_restrict ETH # with ros_management_tools
```

Fine-tuning discovery: ROS_AUTOMATIC_DISCOVERY_RANGE

- since Iron

```
 SUBNET # default: any reachable node
 LOCALHOST # deprecates ROS_LOCALHOST_ONLY
 OFF # no auto-discovery
```

- Discovery is not connection!
  - A node may wait to be discovered by another one

Other tools from `ros_management_tools`

```
ros2cd <package> # jumps to package directory

colbuild # colcon build --symlink-install --continue-on-error
         # also works from anywhere inside the workspace
colbuild -p <packages> # same as --packages-select
colbuild -pu <packages> # same as --packages-up-to
colbuild -t or --this # compile the package we are in
```

https://github.com/oKermorgant/ros_management_tools