Robotics for AI

Development lecture 1: Introduction

Table of content:

ROS: Robot Operating System

- A development standard for robotics.
- Commonly used for prototyping, but also used in industry.
- Forms a communication infrastructure
- Large collection of tools, drivers, libraries and algorithms.
- Allows for hardware abstraction.
- Runs on many types of computing devices (PC, Raspberry PI, Arduino etc.)
- We will be using ROS 1; Melodic

ROS: Robot Operating System

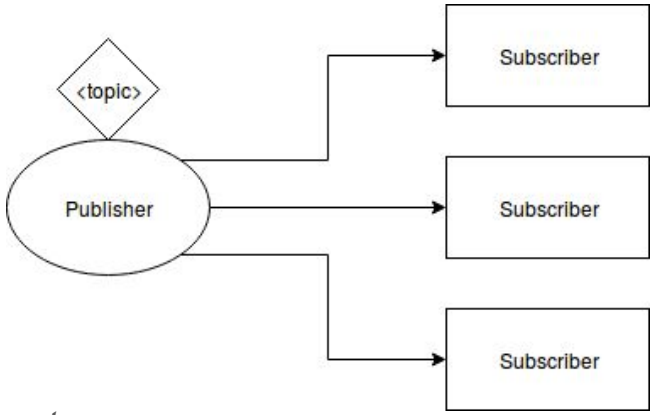
ROS introduction video:

www.ros.org

ROS: Communication structure.

- Every program used in the architecture is a ROS Node
  - *roscore* forms the master node of the network
  - Every ROS node is connected to *roscore*
- Programs can be written in different languages
  - Mostly C++ and Python, but Java and Lisp support exists
- Messages are used to communicate between programs.
  - Messages are specified in ROS as a data structure
  - These messages are compiled to different languages for cross-language support

ROS: Communication structure.



Messaging patterns:

- Data streams: Publishers & Subscribers
  - Publishers *publish* messages of the specified *message type* under the specified *topic name*
  - Subscribers *subscribe* to a topic with the specified *topic name* and *message type*
  - A subscriber implements a *callback function*, which is called when a message is received

Example:

A camera driver implements a *publisher* for a topic *"/front_camera"* to send *Image messages*.

The driver tells *roscore* about the new topic.

A computer vision program can implement a *subscriber* that ask for *Images* from the topic "front_camera"

File: sensor_msgs/Image.msg

Compact Message Definition

```
std_msgs/Header header
uint32 height
uint32 width
string encoding
uint8 is_bigendian
uint32 step
uint8[] data
```

http://docs.ros.org/en/melodic/api/sensor_msgs/html/msg/Image.html

Example:

```python
import rospy
from std_msgs.msg import String

pub = rospy.Publisher('topic_name', String, queue_size=10)
rospy.init_node('node_name')
r = rospy.Rate(10) # 10hz
while not rospy.is_shutdown():
   pub.publish("hello world")
   r.sleep()
```

```python
import rospy
from std_msgs.msg import String

def callback(data):
    rospy.loginfo("I heard %s",data.data)

def listener():
    rospy.init_node('node_name')
    rospy.Subscriber("chatter", String, callback)
    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()
```

ROS: Communication structure.

Messaging patterns:

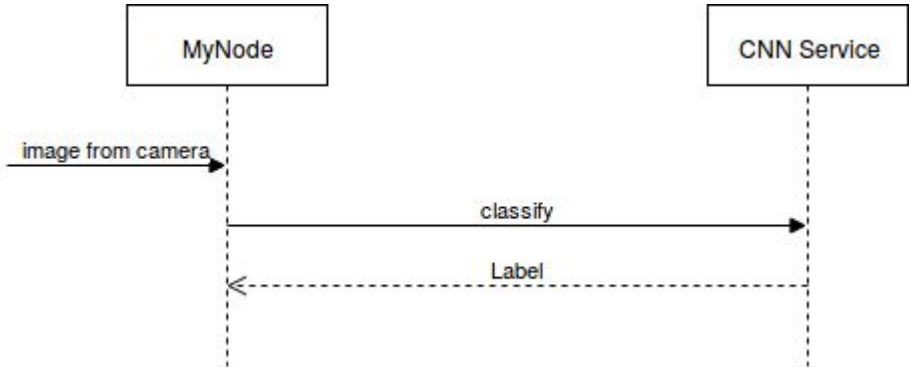
- Data streams: Publishers & Subscribers
- Remote procedure calls: Services
  - A function implemented by another program
  - Offers a *service* to the network
  - A *service* is defined by a *service file* containing:
    - a *request:*
      - a data structure containing the input arguments
    - a *response:*
      - a data structure containing the result/'return' data

ROS: Communication structure.

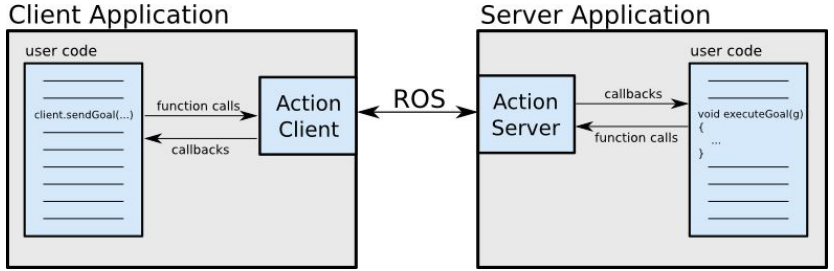
Messaging patterns:

- Data streams: Publishers & Subscribers
- Remote procedure calls: Services
  - An instance of a service uses a *service name* and a service specification.
  - A different program can call this service, as if it is a function.
  - Calling a service is a *blocking* operation.

Example



ROS: Communication structure.



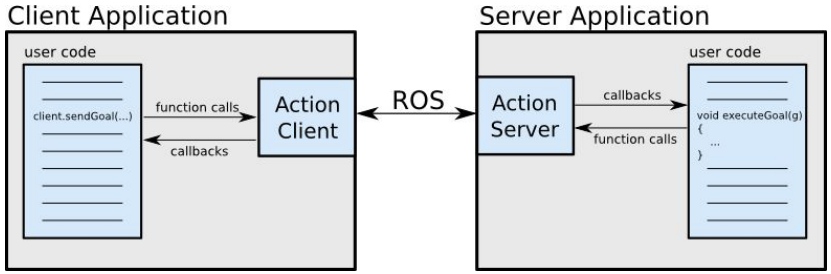
Messaging patterns:

- Data streams: Publishers & Subscribers
- Remote procedure calls: Services
- Actions
  - An action is a ‘task’ to be executed
  - A *client - server* setup
  - The client requests the server to execute the action by sending a “*goal*”.
  - The server executes the action, provides *feedback* about the progress and returns a *result* when the action succeeded or failed.
  - The client’s code is non-blocking. The clients should check every iteration whether the server is finished.

ROS: Communication structure.

Action example: Navigation

- Goal: The target location for the robot
- Feedback: The current position of the robot
- Result: Either SUCCEEDED or ABORTED



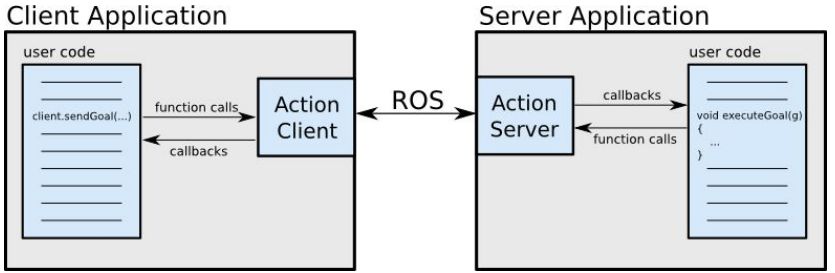
ROS: Communication structure.

Action example: Navigation

Package: Navigation

- Move_base (An Action Server)
  - Can give a goal to navigate to / cancel the goal
  - Gives feedback as the current position of the robot in the world
  - Publisher: Sending command velocities to the base (geometry_msgs/Twist)
  - Services: Make a plan without driving, Clear costmaps

ROS: Communication structure.



Messaging patterns:

- Data streams: Publishers & Subscribers
- Remote procedure calls: Services
- Actions
    - An *action* is specified by an *goal*, *response* and a *results.*
    - An *action client* asks an *action server* to perform a task with an action goal.
    - During the execution, the action server informs the client on the progress with a *response*.
    - Finally the action server gives the *result* to the client.
    - An action call is *non-blocking* when executing tasks

Action example

```python
#! /usr/bin/env python

import roslib
import rospy
import actionlib

from chores.msg import DoDishesAction

class DoDishesServer:
  def __init__(self):
    self.server = actionlib.SimpleActionServer('do_dishes', DoDishesAction, self.execute, False)
    self.server.start()

  def execute(self, goal):
    # Do lots of awesome groundbreaking robot stuff here
    self.server.set_succeeded()


if __name__ == '__main__':
  rospy.init_node('do_dishes_server')
  server = DoDishesServer()
  rospy.spin()
```

```python
#! /usr/bin/env python

import roslib
import rospy
import actionlib

from chores.msg import DoDishesAction, DoDishesGoal

if __name__ == '__main__':
    rospy.init_node('do_dishes_client')
    client = actionlib.SimpleActionClient('do_dishes', DoDishesAction)
    client.wait_for_server()

    goal = DoDishesGoal()
    # Fill in the goal data structure
    goal.dishwasher_id = 10
    client.send_goal(goal)
    waiting = True
    while waiting:
        server_state = client.get_state()
        if server_state == actionlib.GoalStatus.SUCCEEDED:
            print "Success!"
            result = client.get_result()
            waiting = False
        elif server_state == actionlib.GoalStatus.ABORTED:
            print "Failed..."
            waiting = False
```

```
# Define the goal
uint32 dishwasher_id  # Specify which dishwasher we want to use
---
# Define the result
uint32 total_dishes_cleaned
---
# Define a feedback message
float32 percent_complete
```

ROS: Communication structure.

Messages:

- Data structure
- Used for communication between nodes
  - Streaming sensor data
  - Command actuators
- Many useful messages already exist, though you can define your own.
  - Standard primitive types (Integer, Floating point, boolean, etc)
  - Arrays
  - Twist, Odometry, LaserScan

ROS: Communication structure.

Messages: a Twist Message

geometry_msgs/Twist Message

File: geometry_msgs/Twist.msg

Raw Message Definition

```
# This expresses velocity in free space broken into its linear and angular parts.
Vector3  linear
Vector3  angular
```

Compact Message Definition

```
geometry_msgs/Vector3 linear
geometry_msgs/Vector3 angular
```

geometry_msgs/Vector3 Message

File: geometry_msgs/Vector3.msg

Raw Message Definition

```
# This represents a vector in free space.
# It is only meant to represent a direction. Therefore, it does not
# make sense to apply a translation to it (e.g., when applying a
# generic rigid transformation to a Vector3, tf2 will only apply the
# rotation). If you want your data to be translatable too, use the
# geometry_msgs/Point message instead.

float64 x
float64 y
float64 z
```

Compact Message Definition

```
float64 x
float64 y
float64 z
```

How to run ROS nodes

- roscore (starts master server so all nodes can find each other)
- rosrun <package_name> <node_name>
  - runs a single node, possibly with parameters
- python <python_file>
  - starts a python program
- roslaunch <package_name> <launch file>
  - starts a collection of programs and/or parameters to run (a) node(s)

Behaviour programming

Outline

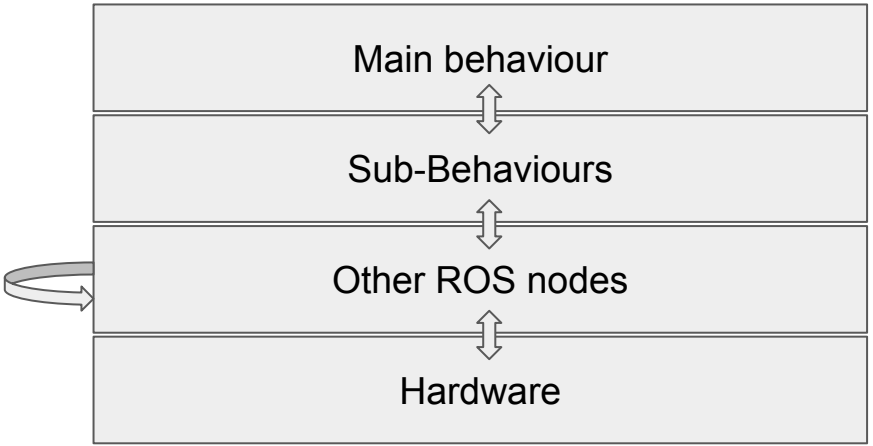
- What is a behaviour?
- Main behaviour & sub behaviours
- Generating and writing behaviours
- Running a behaviour

Behaviour programming

What is a behaviour?

- Top-level code that communicates with other ROS nodes
- Written in Python
- Hierarchical model
- Responsible for fallbacks/recoveries
  - If something fails, make sure to handle that failure.
  - Failures don’t have to be a problem!
  - E.g. if the robot is stuck during navigation, return to an earlier waypoint.

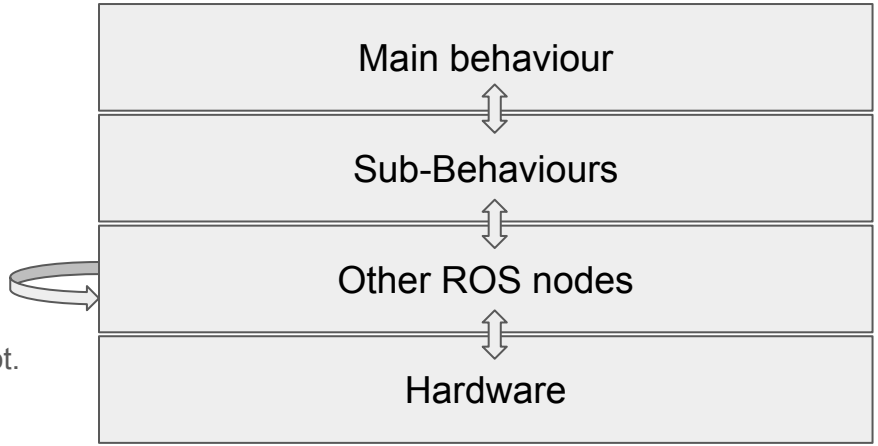
Behaviour programming



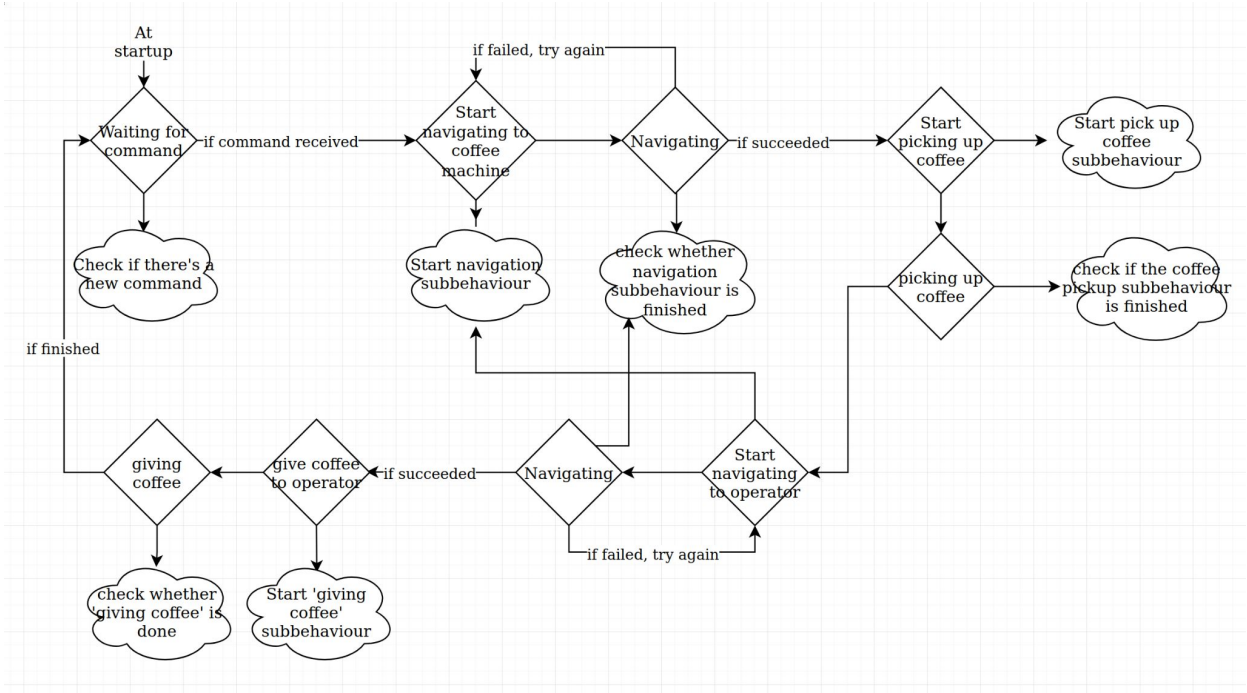
Behaviour programming

The main behaviour:

- The top-level logic defining the current behaviour of the robot.
- Implemented as a *finite state machine.*
- The main behaviour interacts with subbehaviours, which implement specific tasks.



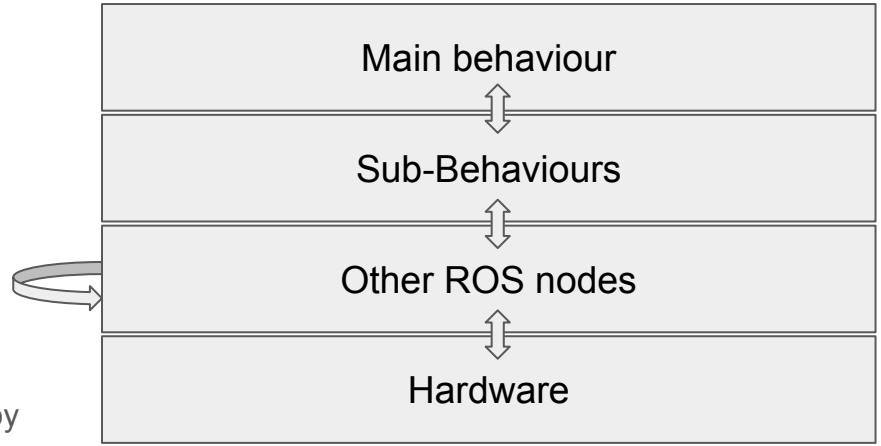
Example: A main behaviour state machine



Behaviour programming

Subbehaviours:

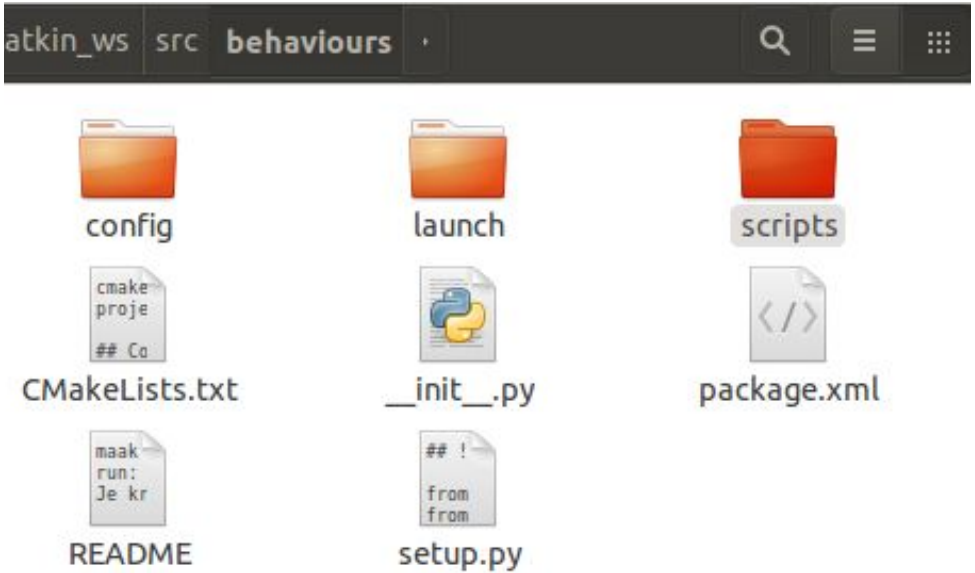
- Implements a specific tasks executions.
- The main behaviour can start a subbehaviour
- Ones started, the subbehaviour start executing the task by interacting with other ROS nodes.
- At the end, a subbehaviour either finishes or fails



Behaviour programming

The behaviour module

First, define the behaviours in a .yaml file in the config folder



Behaviour programming

```
1   main_behaviour: MainBehaviour
2   sub_behaviours:
3     - Random
4     - PrintRandomNumber
```

The behaviour module

First, define the behaviours in a .yaml file in the config folder

For this example: random_behaviour.yaml

Define the main behaviour name and subbehaviour names

Behaviour programming

Next, generate the behavior templates:

```
1   main_behaviour: MainBehaviour
2   sub_behaviours:
3     - Random
4     - PrintRandomNumber
```

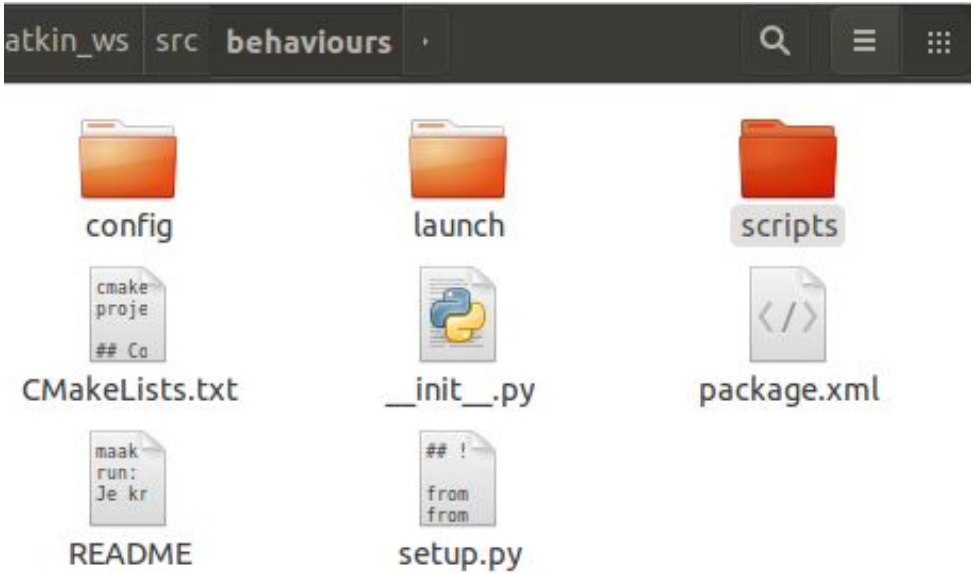
In terminal:

roslaunch behaviour create_behaviours.launch file:=random_behaviour

(Change ‘random_behaviour’ to the name of your yaml file, without ‘.yaml’)

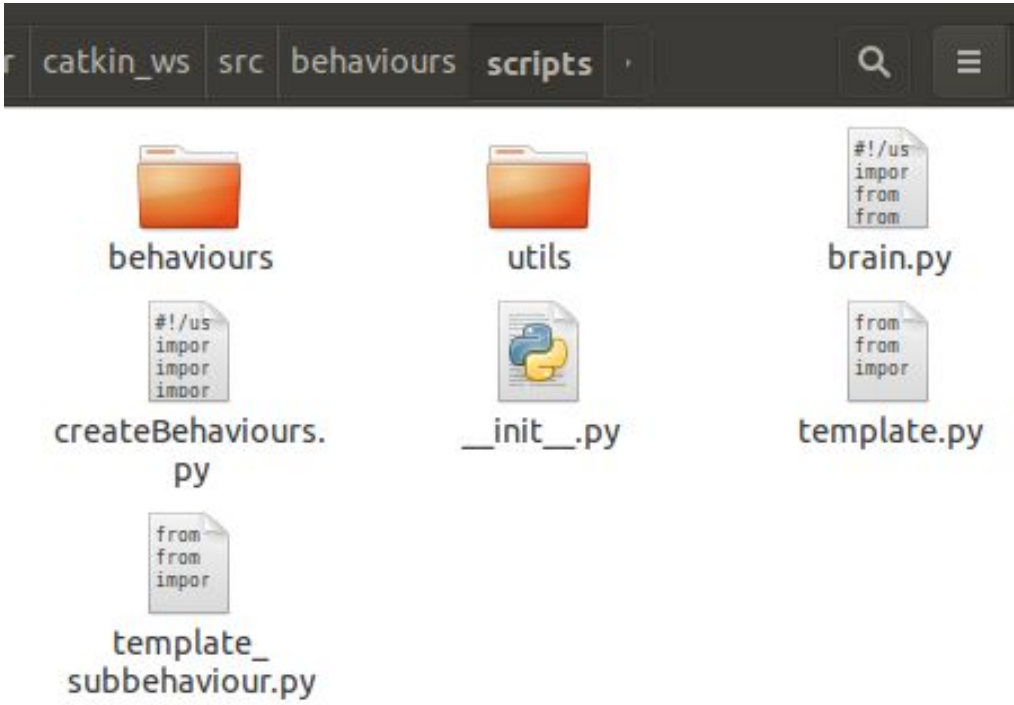
Behaviour programming

Next, we'll take a look at the scripts folder.



Behaviour programming

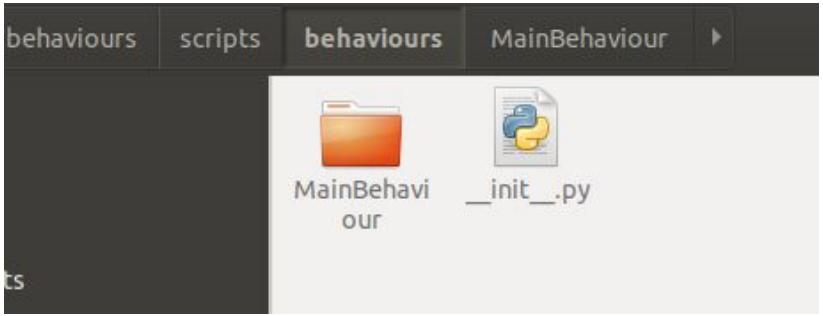
Here, we find the behaviours in the *behaviours* folder

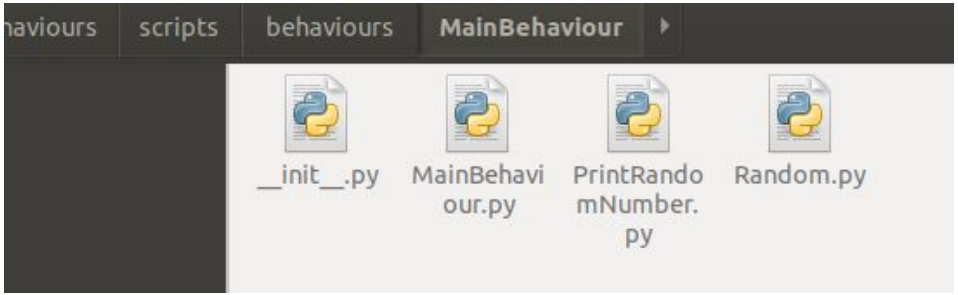


Behaviour programming

```
1   main_behaviour: MainBehaviour
2   sub_behaviours:
3     - Random
4     - PrintRandomNumber
```

Here, we find the behaviours in the *behaviours* folder

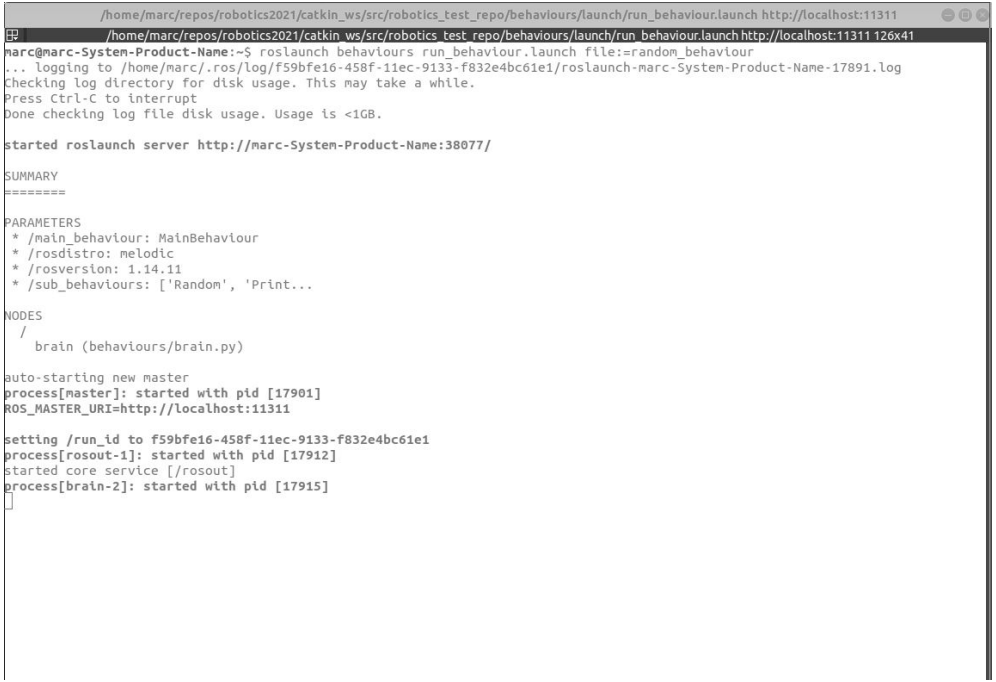




Behaviour programming

When running, it starts out like this:

After this you'll see the behaviour prints, but the behaviours are still empty!



Behaviour programming (Main behaviour)

```
1   from utils.state import State
2   from utils.abstractBehaviour import AbstractBehaviour
3   import rospy
4
5
6   class MainBehaviour(AbstractBehaviour):
7
8       def init(self):
9           pass
10
11      def update(self):
12          pass
```

Behaviour programming (sub behaviour)

```
1   from utils.state import State
2   from utils.abstractBehaviour import AbstractBehaviour
3   import rospy
4
5
6   class Random(AbstractBehaviour):
7
8       def init(self):
9           pass
10
11      def update(self):
12          pass
13
14      def reset(self):
15          self.state = State.idle
16          self.init()
```

Behaviour programming (states)

- idle: Default state before behaviour starts
- start: Default state when running
- finished: Stops the behaviour
- failed: Stops the behaviour

- The behaviour will be running at ‘start’, and all your custom states.

Behaviour programming (states)

```
1   from enum import Enum, unique
2
3   @unique
4   class State(Enum):
5       idle = 0
6       start = 1
7       finished = 2
8       failed = 3
```

This file can be found at ‘behaviours/scripts/utils/state.py’

Behaviour programming (Base class)

```python
from state import State


class AbstractBehaviour(object):

    def __init__(self, all_behaviours):
        self.all_behaviours = all_behaviours
        self.state = State.idle
        self.failureReason = ""

    def get_behaviour(self, name):
        return self.all_behaviours.get_behaviour(name)

    def get_state(self):
        return self.state

    def set_state(self, state):
        self.state = state

    def stopped(self):
        return self.state == State.failed or self.state == State.finished

    def finished(self):
        return self.state == State.finished

    def failed(self):
        return self.state == State.failed

    def get_failure_reason(self):
        return self.failureReason

    def finish(self):
        self.state = State.finished

    def fail(self, reason):
        self.state = State.failed
        self.failureReason = reason

    def start(self):
        self.state = State.start

```

This file can be found at ‘behaviours/scripts/utils/abstractBehaviour.py’

Behaviour programming (states)

- idle: Default state before behaviour starts
- start: Default state when running
- finished: Stops the behaviour
- failed: Stops the behaviour

- The behaviour will be running at ‘start’, and all your custom states.

Behaviour programming

```python
from utils.state import State
from utils.abstractBehaviour import AbstractBehaviour
import rospy
import random

class Random(AbstractBehaviour):

    def init(self):
        self.counter = 0
        self.number = 0

    def update(self):
        if self.counter == 10:
            self.number = random.randint(1,100)
            self.finish()
        self.counter += 1

    def random_number(self):
        return self.number

    def reset(self):
        self.state = State.idle
        self.init()
```

Behaviour programming

```
1  from utils.state import State
2  from utils.abstractBehaviour import AbstractBehaviour
3  import rospy
4  import random
5
6
7  class PrintRandomNumber(AbstractBehaviour):
8
9      def init(self):
10         self.number = 0
11
12     def update(self):
13         random_int = random.randint(0,100)
14         if random_int < 30:
15             self.fail("A descriptive failure reason")
16         elif random_int > 90:
17             print "Printing subbehaviour: I received the random number ", self.number
18             self.finish()
19
20
21     def start(self, random_number):
22         self.state = State.start
23         self.number = random_number
24
25     def reset(self):
26         self.state = State.idle
27         self.init()
```

Behaviour programming

```python
from enum import Enum, unique

@unique
class State(Enum):
    idle = 0
    start = 1
    finished = 2
    failed = 3
    waiting_for_random_number = 4
    waiting_for_printing = 5

```

```python
from utils.state import State
from utils.abstractBehaviour import AbstractBehaviour
import rospy


class MainBehaviour(AbstractBehaviour):

    def init(self):
        self.latest_random_number = 0
        self.random_subbehaviour = self.get_behaviour("Random")
        self.printing_subbehaviour = self.get_behaviour("PrintRandomNumber")

    def update(self):

        if self.state == State.start:
            self.state = State.waiting_for_random_number
            self.random_subbehaviour.start()

        elif self.state == State.waiting_for_random_number:
            if self.random_subbehaviour.stopped():
                if self.random_subbehaviour.finished():
                    print "Got a random number!"
                    self.latest_random_number = self.random_subbehaviour.random_number()
                    self.printing_subbehaviour.start(self.latest_random_number)
                    self.state = State.waiting_for_printing
                elif self.random_subbehaviour.failed():
                    print "Failed getting random number.. Restarting.."
                    self.state = State.start

        elif self.state == State.waiting_for_printing:
            if self.printing_subbehaviour.stopped():
                if self.printing_subbehaviour.finished():
                    print "Finished printing the random number! Stopping main behaviour.."
                    self.finish()
                elif self.printing_subbehaviour.failed():
                    print "failed printing the number.. Trying again.."
                    self.printing_subbehaviour.start(self.latest_random_number)

```

Output

```
marc@marc-System-Product-Name:~$ roslaunch behaviours run_behaviour.launch file:=random_behaviour
... logging to /home/marc/.ros/log/04c15afc-4596-11ec-9133-f832e4bc61e1/roslaunch-marc-System-Product-Name-19295.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://marc-System-Product-Name:37259/

SUMMARY
========

PARAMETERS
 * /main_behaviour: MainBehaviour
 * /rosdistro: melodic
 * /rosversion: 1.14.11
 * /sub_behaviours: ['Random', 'Print...

NODES
  /
    brain (behaviours/brain.py)

auto-starting new master
process[master]: started with pid [19305]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to 04c15afc-4596-11ec-9133-f832e4bc61e1
process[rosout-1]: started with pid [19316]
started core service [/rosout]
process[brain-2]: started with pid [19319]
Got a random number!
failed printing the number.. Trying again..
failed printing the number.. Trying again..
failed printing the number.. Trying again..
failed printing the number.. Trying again..
failed printing the number.. Trying again..
Printing subbehaviour: I received the random number  94
Finished printing the random number! Stopping main behaviour..
================================================================================REQUIRED process [brain-2] has died!
process has finished cleanly
log file: /home/marc/.ros/log/04c15afc-4596-11ec-9133-f832e4bc61e1/brain-2*.log
Initiating shutdown!
================================================================================
[brain-2] killing on exit
[rosout-1] killing on exit
[master] killing on exit
shutting down processing monitor...
... shutting down processing monitor complete
done
```

Behaviour programming

- Use clear coding style!!
- No variable names with less than 3 characters!
- Use descriptive variable names
- Consider all variables in subbehaviours *private*.
  - Add new member functions to subbehaviours if needed.
- Functions should do as they’re named, not more, not less!
- If we can’t read your code, we can’t help you with debugging…..

Python

Be sure you know to work with: (https://www.tutorialspoint.com/python/)

- basic syntax
- variable types
- basic operators
- decision making
- loops
- lists
- dictionaries
- functions
- classes / objects

Git

Be sure you know how to: (https://try.github.io)

- clone
- commit
- push
- pull
- merge
- create a new branch

At first usage on the VM / PC, you'll have to configure your name & email:

git config --global user.name "*My Name*"

git config --global user.email *"my@email.com*"

Git

Since you'll be working (remotely) in pairs, merge conflicts are likely.

Clearly communicate between each other about resolving the conflicts.


That's all Folks!

Python

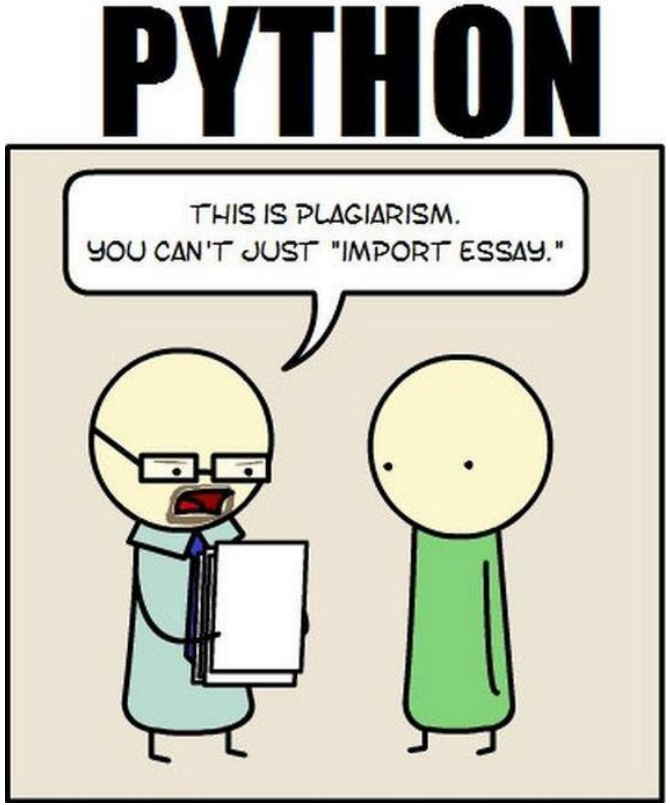
Why Python?

Basic Types

More advanced types

Flow Control

Classes



Python

Why Python?

- Easy to read/write
- Lots of compatible libraries (often implemented in C++/Fortran)
- Write powerful high level code, though it’s a bit slow.. So, keep it simple and use library functions as much as possible.

Python

Basic types & Functions

- Types are implicit!
- int, float, string ….
- Function declaration with ‘def name(args)’

slide1.py > ...

```
1   def add(a, b): # a + b
2       return a + b
3
4   def mod(a, b):
5       ans = a
6       while ans < 0:
7           ans += b
8       while ans > b:
9           ans -= b
10      return ans
11
12  #using ints
13  a = 42
14  b = 16
15  print('ints:')
16  print('a: ', a, ' b: ', b, " sum = ", add(a,b))
17  print('a mod b = ', mod(a,b))
```

Python

Basic types & Functions

- Types are implicit!
- int, float, string ….
- Function declaration with ‘def name(args)’

slide1.py > ...

```
1  def add(a, b): # a + b
2      return a + b
3
4  def mod(a, b):
5      ans = a
6      while ans < 0:
7          ans += b
8      while ans > b:
9          ans -= b
10     return ans
11
12 #using ints
13 a = 42
14 b = 16
15 print('ints:')
   print('a: ', a, ' b: ', b, " sum = ", add(a,b))
   print('a mod b = ', mod(a,b))
```

```
ints:
a:  42  b:  16  sum =  58
a mod b =  10
floats:
a:  42.42  b:  16.4  sum =  58.82
```

Python

More advanced types

- Dictionaries: Key + value storage
- Lists: A list of anything you want
- Tuples: A set of 2 variables

slide2.py > ...

```
1  #using lists
2  a = [16, 3, 12, 14, 1]
3  for num in a:
4      print(num)
5
6  #using dictionaries
7  basket = {'apples' : 10, 'coffee' : 3, 'chocolate' : 1}
8  print(basket['apples'], ' apples')
9  for key in basket.keys():
10     print(key, ': ', basket[key])
11
12 #using tuples
13 a = ('door', 'house')
14 b = ('car', 'wheel')
15 c = ('ans', 42)
16 print(a, b, c)
17 print(c[0], c[1])
```

Python

More advanced types

- Dictionaries: Key + value storage
- Lists: A list of anything you want
- Tuples: A set of 2 variables

slide2.py > ...

```
1  #using lists
2  a = [16, 3, 12, 14, 1]
3  for num in a:
4      print(num)
5
6  #using dictionaries
7  basket = {'apples' : 10, 'coffee' : 3, 'chocolate' : 1}
8  print(basket['apples'], ' apples')
9  for key in basket.keys():
10     print(key, ': ', basket[key])
11
12 #using tuples
13 a = ('door', 'house')
14 b = ('car', 'wheel')
15 c = ('ans', 42)
16 print(a, b, c)
17 print(c[0], c[1])
```

```
16
3
12
14
1
10  apples
coffee :  3
apples :  10
chocolate :  1
('door', 'house') ('car', 'wheel') ('ans', 42)
ans 42
```

Python

Flow Control

- Scopes are indicated using indentation. Watch out for equal length Tabs / Spaces!

slide3.py > ...

```
1   ##approximating the golden ratio using Fibonacci:
2   a = 1
3   b = 1
4   print(a)
5   print(b)
6
7   for idx in range(12):
8       c = a
9       a = add(a, b)
10      b = c
11      print('phi = ', float(b) / a)
12
13  a = 0
14  while(a < 10):
15      a += 2
16      if a == 4:
17          break;
18
```

Python

Flow Control

- Scopes are indicated using indentation. Watch out for equal length Tabs / Spaces!

```
1 1
2 1
3 phi =  0.5
4 phi =  0.666666666667
5 phi =  0.6
6 phi =  0.625
7 phi =  0.615384615385
8 phi =  0.619047619048
9 phi =  0.617647058824
10 phi =  0.618181818182
11 phi =  0.61797752809
12 phi =  0.618055555556
13 phi =  0.618025751073
14 phi =  0.618037135279
```

slide3.py > ...

```
1  ##approximating the golden ratio using Fibonacci:
2  a = 1
3  b = 1
4  print(a)
5  print(b)
6
7  for idx in range(12):
8      c = a
9      a = add(a, b)
10     b = c
11     print('phi = ', float(b) / a)
12
13 a = 0
14 while(a < 10):
15     a += 2
16     if a == 4:
17         break;
18
```

Python

Classes

- A class is a description of a set of variables and functions.
- An object in an individual instance of such a class.
- “self” refers to the object.
- __init__(self) is the constructor.

dog.py

```python
class Dog:

    def __init__(self, name, age):
        self.name = name
        self.age = age
        self.hunger = 0

    def get_name(self):
        return self.name

    def rename(self, name):
        self.name = name

    def incr_age(self):
        self.age += 1

    def get_age(self):
        return self.age

if __name__ == "__main__":
    dog = Dog("frodo", 2)
    print(dog.get_name())
    dog.rename("sam")
    print(dog.get_name())
    dog.incr_age()
    print(dog.get_age())
```

Python

Classes

- “self” refers to the object.
- __init__(self) is the constructor.

```
marc@Marc-Workstation:~/repos/oop_demo$ python3 dog.py
frodo
sam
3
```

dog.py

dog.py > Dog

```python
class Dog:

    def __init__(self, name, age):
        self.name = name
        self.age = age
        self.hunger = 0

    def get_name(self):
        return self.name

    def rename(self, name):
        self.name = name

    def incr_age(self):
        self.age += 1

    def get_age(self):
        return self.age

if __name__ == "__main__":
    dog = Dog("frodo", 2)
    print(dog.get_name())
    dog.rename("sam")
    print(dog.get_name())
    dog.incr_age()
    print(dog.get_age())
```

Git

- Why Git?
- Overview
- Terminology
- Example 0: Adding and Removing files
- Example 1: Pulling and Pushing Changes
- Example 2: Merging
- Example 3: Branching

Git

Why Git?

- Version Control
- Working on files together, while both on own computer
- Acts as logbook, over entire project.
- Allows going back to earlier states.
- Once you learned Git well, you never go back..

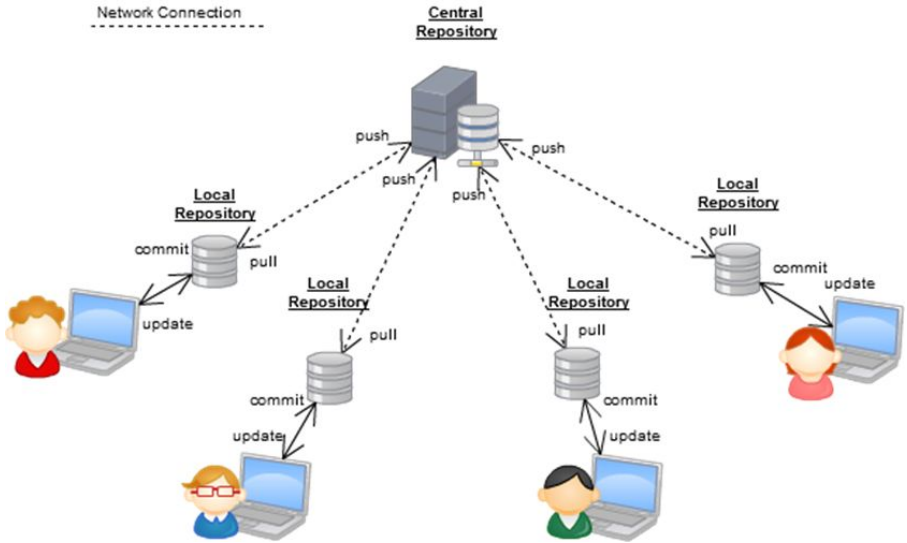
Git

Overview:

A central server maintains the global state.

Users have a local state, which they can edit.

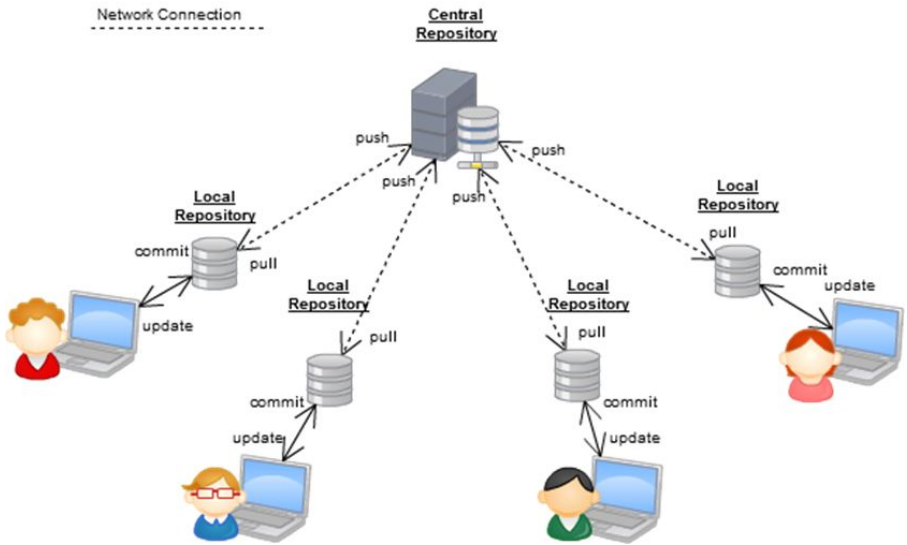
Users can sync. their state with the central server.



Git

Terminology:

- Commit: Make a new state with your recent changes and log message..
- Pull: Request changes from CR.
- Push: Sending new state to CR (Pull first).
- Merge: Merging your state another commit.
- Branch: Create a new concurrent version of your current branch. (default: “master”)



Git

Example 0: Adding / Removing files

- First we add the new file
- Then we check the git status for listed changes.
- We commit our changes, with a message.
- Pull possible changes from CR
- Push changes to CR.

```
marc@nozecvault:~/repos/tutorial$ echo "Hello, World!" >> myfile.txt
marc@nozecvault:~/repos/tutorial$ git add myfile.txt
marc@nozecvault:~/repos/tutorial$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   myfile.txt

marc@nozecvault:~/repos/tutorial$ git commit -am "added myfile.txt"
[master 2747f6d] added myfile.txt
 1 file changed, 1 insertion(+)
 create mode 100644 myfile.txt
marc@nozecvault:~/repos/tutorial$ git pull
Already up-to-date.
marc@nozecvault:~/repos/tutorial$ git push
Counting objects: 4, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 295 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To git@github.com:MarcGroef/tutorial.git
   91739e3..2747f6d  master -> master
marc@nozecvault:~/repos/tutorial$
```

Git

Example 0: Adding / Removing files

- git rm also removes the actual file.
- To keep it local: --cached flag
- To force removal: -f

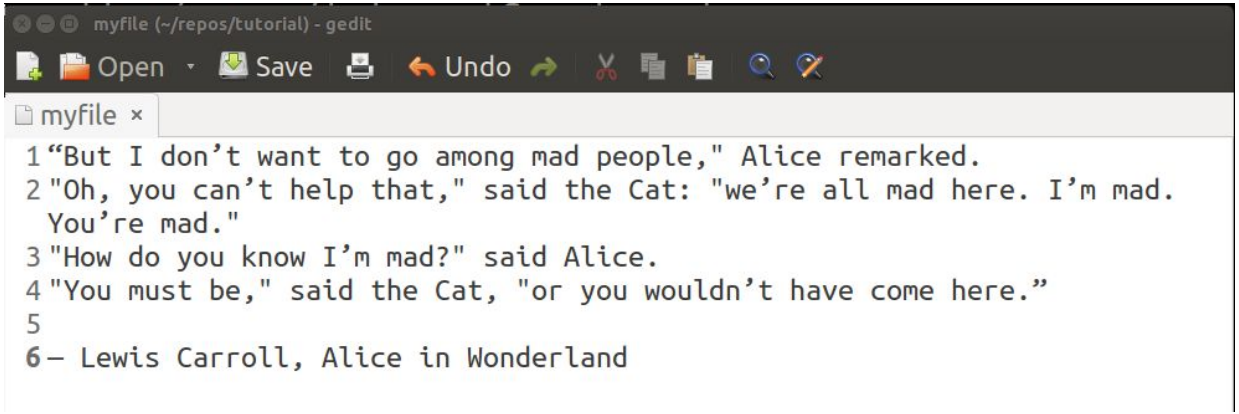
```
marc@nozecvault:~/repos/tutorial$ ls
myfile.txt  readme
marc@nozecvault:~/repos/tutorial$ git rm myfile.txt
rm 'myfile.txt'
marc@nozecvault:~/repos/tutorial$ ls
readme
marc@nozecvault:~/repos/tutorial$ git commit -am "removed myfile.txt"
[master f6762ba] removed myfile.txt
 1 file changed, 1 deletion(-)
 delete mode 100644 myfile.txt
marc@nozecvault:~/repos/tutorial$ git pull
Already up-to-date.
marc@nozecvault:~/repos/tutorial$ git push
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (1/1), done.
Writing objects: 100% (2/2), 238 bytes | 0 bytes/s, done.
Total 2 (delta 0), reused 0 (delta 0)
To git@github.com:MarcGroef/tutorial.git
   2747f6d..f6762ba  master -> master
```

Introducing: Alice and Bob

Git

Example 1: Pulling and Pushing

- First Alice writes some content, which she commits and pushes.



Git

Example 1: Pulling and Pushing

- First Alice writes some content, which she commits and pushes.

```
marc@nozecvault:~/repos/tutorial$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   myfile

no changes added to commit (use "git add" and/or "git commit -a")
marc@nozecvault:~/repos/tutorial$ git commit -am "Alice: Wrote something"
[master 4dcf769] Alice: Wrote something
 1 file changed, 6 insertions(+)
marc@nozecvault:~/repos/tutorial$ git push
Counting objects: 5, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 489 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To git@github.com:MarcGroef/tutorial.git
   29ea1e9..4dcf769  master -> master
```

Git

Example 1: Pulling and Pushing

- Next, Bob pulls the changes


```
marc@nozecvault:~/repos/tutorial2$ ls
myfile  readme
marc@nozecvault:~/repos/tutorial2$ cat myfile
marc@nozecvault:~/repos/tutorial2$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

nothing to commit, working directory clean
marc@nozecvault:~/repos/tutorial2$ git pull
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From github.com:MarcGroef/tutorial
   29ea1e9..4dcf769  master     -> origin/master
Updating 29ea1e9..4dcf769
Fast-forward
 myfile | 6 ++++++
 1 file changed, 6 insertions(+)
marc@nozecvault:~/repos/tutorial2$ cat myfile
“But I don’t want to go among mad people," Alice remarked.
"Oh, you can’t help that," said the Cat: "we’re all mad here. I’m mad. You’re mad."
"How do you know I’m mad?" said Alice.
"You must be," said the Cat, "or you wouldn’t have come here.”

– Lewis Carroll, Alice in Wonderland
marc@nozecvault:~/repos/tutorial2$
```

Git

Example 1: Pulling and Pushing

- Then Bob changes “the Cat” to “R.B.T.” ...

myfile ×

```
1 “But I don’t want to go among mad people," Alice remarked.
2 "Oh, you can’t help that," said R.B.T.: "we’re all mad here.
  I’m mad. You’re mad."
3 "How do you know I’m mad?" said Alice.
4 "You must be," said R.B.T., "or you wouldn’t have come here.”
5
6 – Lewis Carroll, Alice in Wonderland, edited by Bob
```

Git

Example 1: Pulling and Pushing

- … and Bob commits and pushes.


```
marc@nozecvault:~/repos/tutorial2$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   myfile

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        myfile~

no changes added to commit (use "git add" and/or "git commit -a")
marc@nozecvault:~/repos/tutorial2$ git commit -am "Bob: changed the Cat to RBT"
[master 863ff3b] Bob: changed the Cat to RBT
 1 file changed, 3 insertions(+), 3 deletions(-)
marc@nozecvault:~/repos/tutorial2$ git push
Counting objects: 5, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 347 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To git@github.com:MarcGroef/tutorial.git
   4dcf769..863ff3b  master -> master
marc@nozecvault:~/repos/tutorial2$
```

Sudo →

Git

Example 1: Pulling and Pushing

- In the meanwhile, Alice was making her own changes:
  - She replaces “Alice” with “Sudo”
  - ….

```
1 “But I don’t want to go among mad people," Sudo remarked.
2 "Oh, you can’t help that," said the Cat: "we’re all mad here. I’m mad. You’re mad."
3 "How do you know I’m mad?" said Sudo.
4 "You must be," said the Cat, "or you wouldn’t have come here.”
5
6 – Lewis Carroll, Alice in Wonderland, modified by Alice
```

Git

Example 2: Merging

- And she commits, pulls…..
- A merge conflict occurs!

```
marc@nozecvault:~/repos/tutorial$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   myfile

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        myfile~

no changes added to commit (use "git add" and/or "git commit -a")
marc@nozecvault:~/repos/tutorial$ git commit -am "changed Alice to Sudo"
[master 189e1bd] changed Alice to Sudo
 1 file changed, 3 insertions(+), 3 deletions(-)
marc@nozecvault:~/repos/tutorial$ git pull
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 3 (delta 1), pack-reused 0
Unpacking objects: 100% (3/3), done.
From github.com:MarcGroef/tutorial
   4dcf769..863ff3b  master     -> origin/master
Auto-merging myfile
CONFLICT (content): Merge conflict in myfile
Automatic merge failed; fix conflicts and then commit the result.
marc@nozecvault:~/repos/tutorial$
```

Git

Example 2: Merging (Merge conflict)

- First she checks which files contain conflicts, with ‘git commit -a’.
  - While solving conflicts, don’t close this terminal!

```
  GNU nano 2.2.6    File: /home/marc/repos/tutorial/.git/COMMIT_EDITMSG

Merge branch 'master' of github.com:MarcGroef/tutorial

Conflicts:
        myfile
#
# It looks like you may be committing a merge.
# If this is not correct, please remove the file
#       .git/MERGE_HEAD
# and try again.


# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Your branch and 'origin/master' have diverged,
# and have 1 and 1 different commit each, respectively.
#   (use "git pull" to merge the remote branch into yours)
#
# All conflicts fixed but you are still merging.
#
# Changes to be committed:
#       modified:   myfile
#
# Untracked files:
#       myfile~
#
                                [ Read 26 lines ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text ^T To Spell
```

Git

Example 2: Merging (Merge conflict)

- To resolve the conflict, she opens the respective file...

```
<<<<<<< HEAD
“But I don’t want to go among mad people," Sudo remarked.
"Oh, you can’t help that," said the Cat: "we’re all mad here. I’m mad. You’re mad.'
"How do you know I’m mad?" said Sudo.
"You must be," said the Cat, "or you wouldn’t have come here.”

– Lewis Carroll, Alice in Wonderland, modified by Alice
=======
“But I don’t want to go among mad people," Alice remarked.
"Oh, you can’t help that," said R.B.T.: "we’re all mad here. I’m mad. You’re mad."
"How do you know I’m mad?" said Alice.
"You must be," said R.B.T., "or you wouldn’t have come here.”

– Lewis Carroll, Alice in Wonderland, edited by Bob
>>>>>>> 863ff3bbc93c7c1d8ed6697e508db80bd333dc39
```

Git

Example 2: Merging (Merge conflict)

- … fixes the conflict ….

myfile ×

```
1 “But I don’t want to go among mad people," Sudo remarked.
2 "Oh, you can’t help that," said R.B.T.: "we’re all mad here. I’m mad. You’re mad."
3 "How do you know I’m mad?" said Sudo.
4 "You must be," said R.B.T., "or you wouldn’t have come here.”
5
6 – Lewis Carroll, Alice in Wonderland, modified by Alice and Bob
```

Git

Example 2: Merging (Merge conflict)

- and finally commits her changes.
- close with ctrl + X , Y then Enter

```
  GNU nano 2.2.6    File: /home/marc/repos/tutorial/.git/COMMIT_EDITMSG     Modified

Fixed conflict ⟸
Merge branch 'master' of github.com:MarcGroef/tutorial

Conflicts:
        myfile
#
# It looks like you may be committing a merge.
# If this is not correct, please remove the file
#       .git/MERGE_HEAD
# and try again.


# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Your branch and 'origin/master' have diverged,
# and have 1 and 1 different commit each, respectively.
#   (use "git pull" to merge the remote branch into yours)
#
# All conflicts fixed but you are still merging.
#
# Changes to be committed:
#       modified:   myfile
#
# Untracked files:
#       myfile~

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text ^T To Spell
```

Git

Example 2: Merging (Merge conflict)

- And pushes the fix the the CR

```
marc@nozecvault:~/repos/tutorial$ git pull
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 3 (delta 1), pack-reused 0
[master 3cb61ae] Fixed conflict Merge branch 'master' of github.com:MarcGroef/tutorial
marc@nozecvault:~/repos/tutorial$ git push
Counting objects: 10, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 922 bytes | 0 bytes/s, done.
Total 6 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), done.
To git@github.com:MarcGroef/tutorial.git
   863ff3b..3cb61ae  master -> master
marc@nozecvault:~/repos/tutorial$
```

Git

Example 2: Merging

Though in practice, Git is often smart enough to merge by itself. :)

Git

Example 3: Branching

Initially there's 1 branch: master

You can branch from your current branch, so you can work on another version.

git checkout -b mynewbranchname

you can switch branches using:

git checkout branchname

```
marc@nozecvault:~/repos/tutorial$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

nothing to commit, working directory clean
marc@nozecvault:~/repos/tutorial$ git checkout -b mynewbranch
Switched to a new branch 'mynewbranch'
marc@nozecvault:~/repos/tutorial$ git status
On branch mynewbranch
nothing to commit, working directory clean
marc@nozecvault:~/repos/tutorial$ git checkout master
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.
marc@nozecvault:~/repos/tutorial$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

nothing to commit, working directory clean
marc@nozecvault:~/repos/tutorial$
```

Git

Example 3: Branching

Merging branches:

On mynewbranch we add a file “otherfile” and commit.

checkout to master

git merge mynewbranch

we receive the “otherfile” in master

```
marc@nozecvault:~/repos/tutorial$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

nothing to commit, working directory clean
marc@nozecvault:~/repos/tutorial$ git checkout -b mynewbranch
Switched to a new branch 'mynewbranch'
marc@nozecvault:~/repos/tutorial$ git status
On branch mynewbranch
nothing to commit, working directory clean
marc@nozecvault:~/repos/tutorial$ git checkout master
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.
marc@nozecvault:~/repos/tutorial$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

nothing to commit, working directory clean
marc@nozecvault:~/repos/tutorial$
```


That's all Folks!