Institut Pasteur C3BI - USR 3756

# BioBlend module, a python library to use Galaxy API

Olivia Doppelt-Azeroual, Fabien Mareuil

## Plan

## Introduction

- The Galaxy API enables developers to access Galaxy functionalities using Python scripts
- BioBlend is a Python overlay implemented to facilitate the writing of those scripts
    - Implemented by Enis Afgane
    - It is available on github: https://github.com/afgane/bioblend and is in the pip packages (pip install bioblend)
    - A complete documentation is available at http://bioblend.readthedocs.io/en/latest/
    - BioBlend enables the manipulation of Galaxy entities (libraries; histories; datasets) as Python Objects

return

## Applications

- On the https://galaxy.pasteur.fr instance, we use BioBlend for several tasks and projects:
    - For Galaxy administration: the automated creation of libraries for new internal users, the groups allocation for new users,...
    - For several project:
        - In ReGaTE, we use BioBlend to retrieve a list of installed tools on a Galaxy instance (article in review in GigaScience)
        - In MetaGenSense (in press), BioBlend is used to mime all Galaxy steps from the upload of big data to the workflow launching and the data results and transfer

return

## Goal

1. Get familiar with BioBlend with ipython
2. Launch a Galaxy job / Visualize your actions with Galaxy
3. Launch a Galaxy workflow / Visualize your actions with Galaxy

## Before we start

1. Authentication for Bioblend - Get your API key:

    - On your Galaxy, click on the User tab and ont the "API Keys" line
    - Click on "Generate a new key now"

2. Install the tools and workflow on your Galaxy:

    1. The tools: Click on the Admin tab
        - In the Tools and Tool Shed category, click on the line **Search Tool Shed**
        - Select the **"Galaxy Main Tool Shed"**, and the **"Browse valid repositories"** line
        - Search and install *bam_to_sam* and *samtools_sort* from IUC owner
    2. The workflow:
        - Get the workflow file (.ga) from https://github.com/fmareuil/formationbioblend
        - Import the workflow in galaxy:
          Click on Workflow tab and "Upload or import workflow" button
    3. Launch ipython on a terminal

## Let's start ...

# Connect with Galaxy using ipython:

- Get your API key and your Galaxy URL
- Import the GalaxyInstance object from BioBlend module:

```
from bioblend.galaxy import GalaxyInstance
```

- Create your GalaxyInstance instance object using your url and your key

```
gi = GalaxyInstance(url="http://127.0.0.1:8080", key="your key")
```

**Why ipython:**

- Automatic completion
  ==> type *gi.* and the tab puis appuyez sur la touche tab key

- To better understand BioBlend methods and classes, you can use *help(command), object??, object?...*

**During all the training, each command results are stored in variables**

# To launch a Galaxy job

- Understanding the **run_tool** method:
    - The help command lets the user know what are the arguments

```
help(gi.tools.run_tool)
```

```
run_tool(self, history_id, tool_id, tool_inputs)
   Runs tool specified by tool_id in history indicated
   by history_id with inputs from dict tool_inputs
   :param history_id: encoded ID of the history in which to run the tool
   :param tool_id: ID of the tool to be run
   :param tool_inputs: dictionary of input datasets and parameters
      for the tool (see below)
   The tool_inputs dict should contain input datasets and parameters
   in the (largely undocumented) format used by the Galaxy API.
```

- To resume, in this first part, we need to retrieve:
    1. A *history_id*, where the input data is and where the output data will be
    2. A *tool_id*, which will tell Galaxy which tool to execute
    3. *tool_inputs*, dictionary storing the data used to run the tool

return

## Histories Object

- Try to get your histories list with BioBlend
- Create a new history (*It will be our work history for this tutorial*)

http://bioblend.readthedocs.org

## Histories Object

- Try to get your histories list with BioBlend
- Create a new history (*It will be our work history for this tutorial*)

http://bioblend.readthedocs.org

```
list_histories = gi.histories.get_histories()

new_history = gi.histories.create_history(name='my_history')
```

- Now that your history is created, you will need to upload some data in it.

## Histories Object

- Try to get your histories list with BioBlend
- Create a new history (*It will be our work history for this tutorial*)

http://bioblend.readthedocs.org

```
list_histories = gi.histories.get_histories()

new_history = gi.histories.create_history(name='my_history')
```

- Now that your history is created, you will need to upload some data in it.

- No BioBlend method to directly upload data from your file system to a history exists, a data can be uploaded in a history from a Galaxy library

```
help(gi.histories.upload_dataset_from_library)
```

return

*history_id*

## Libraries Object

- Check if there is a method to upload a data from your filesystem

```
help(gi.libraries.upload_file_from_local_path)
```

## Libraries Object

- Check if there is a method to upload a data from your filesystem

```
help(gi.libraries.upload_file_from_local_path)
```

- Create a library and set the rights to this library

you will need your role *id*, look for the methods of the Class *gi.roles*

## Libraries Object

- Check if there is a method to upload a data from your filesystem

```
help(gi.libraries.upload_file_from_local_path)
```

- Create a library and set the rights to this library

you will need your role *id*, look for the methods of the Class *gi.roles*

```
role_id = gi.roles.get_roles()[0]['id']
new_lib = gi.libraries.create_library('my_library')
gi.libraries.set_library_permissions(new_lib['id'], access_in=['role_id'],
      modify_in=['role_id'], add_in=['role_id'], manage_in=['role_id'])
```

- Import a BAM file in your library

## Libraries Object

- Check if there is a method to upload a data from your filesystem

```
help(gi.libraries.upload_file_from_local_path)
```

- Create a library and set the rights to this library

you will need your role *id*, look for the methods of the Class *gi.roles*

```
role_id = gi.roles.get_roles()[0]['id']
new_lib = gi.libraries.create_library('my_library')
gi.libraries.set_library_permissions(new_lib['id'], access_in=['role_id'],
      modify_in=['role_id'], add_in=['role_id'], manage_in=['role_id'])
```

- Import a BAM file in your library

```
list_data = gi.libraries.upload_file_from_local_path(new_lib['id'], local_path)
```

- Transfer the BAM file from your library in your new history

# Libraries Object

- Check if there is a method to upload a data from your filesystem

```
help(gi.libraries.upload_file_from_local_path)
```

- Create a library and set the rights to this library

you will need your role *id*, look for the methods of the Class *gi.roles*

```
role_id = gi.roles.get_roles()[0]['id']
new_lib = gi.libraries.create_library('my_library')
gi.libraries.set_library_permissions(new_lib['id'], access_in=['role_id'],
        modify_in=['role_id'], add_in=['role_id'], manage_in=['role_id'])
```

- Import a BAM file in your library

```
list_data = gi.libraries.upload_file_from_local_path(new_lib['id'], local_path)
```

- Transfer the BAM file from your library in your new history

```
data_history = gi.histories.upload_dataset_from_library(new_history['id'],
        list_data[0]['id'])
```

# Tools Object (1/3)

- To run a tool, its 'id' is needed:

```
help(gi.tools.run_tool)
```

- Get the samtools sort tool id

its name is "sort"

*history_id*

*tool_id*

## Tools Object (1/3)

- To run a tool, its 'id' is needed:

```
help(gi.tools.run_tool)
```

- Get the samtools sort tool id

its name is "sort"

```
list_tool = gi.tools.get_tools(name='sort')
```

*history_id*

*tool_id*

*tool_inputs*

## Tools Object (2/3)

- The dictionary *tool_inputs* is needed to run a tool
- It is a python dictionary defined by specific methods from the *bioblend.galaxy.tools.inputs* Class

```
from bioblend.galaxy.tools.inputs import inputs
```

- The *inputs* method instanciates a class called *InputsBuilder*:

```
help(inputs)
```

- Each input from the tool XML needs to be defined using the methods *set_param* or *set_dataset_param* from *InputsBuilder* Class ==> If the input format is "data", the method to use is *set_dataset_param*
- Here is an example:

```
myinputs = inputs().set_param("param1",'value')
      .set_dataset_param("data1",'dataset_id',src="hda")
```

**To run the tool samtools sort, we need more information on the tool itself**

## Tools Object (3/3)

*history_id*

*tool_id*

*tool_inputs*

- Get the details on the "samtool sort" tool

## Tools Object (3/3)

*history_id*

*tool_id*

*tool_inputs*

- Get the details on the "samtool sort" tool

```
detail_tool = gi.tools.show_tool(list_tool[0]['id'],io_details=True)
```

## Tools Object (3/3)

*history_id*

*tool_id*

*tool_inputs*

- Get the details on the "samtool sort" tool

```
detail_tool = gi.tools.show_tool(list_tool[0]['id'],io_details=True)
```

```
detail_tool['inputs']
[{u'argument': None,
   u'edam_formats': [u'format_2572'],
   u'extensions': [u'bam'],
   ...
   u'multiple': False,
   u'name': u'input1', <-------------------------------------|
   u'optional': False,                                        |
   u'options': {u'hda': [], u'hdca': []},                     |
   u'type': u'data'},                                         | Critical
  {...                                                        | information
   u'name': u'sort_mode', <-----------------------------------|
   u'optional': False,                                        |
   u'options': [[u'Chromosomal coordinates', u'', True], <-----|
    [u'Read names', u'-n', False]],
   u'type': u'select',
   u'value': u''}]
```

## Tools Object (3/3)

- Get the details on the "samtool sort" tool

```
detail_tool = gi.tools.show_tool(list_tool[0]['id'],io_details=True)
```

- Inputs information can be displayed with *detail_tool['inputs']*, use this to instantiate the inputs object

*history_id*

*tool_id*

*tool_inputs*

## Tools Object (3/3)

- Get the details on the "samtool sort" tool

```
detail_tool = gi.tools.show_tool(list_tool[0]['id'],io_details=True)
```

- Inputs information can be displayed with detail_tool['inputs'], use this to instantiate the inputs object

```
myinputs = inputs().set_dataset_param("input1", data_history['id'], src='hda')
                .set_param("sort_mode","")
```

## Tools Object (3/3)

*history_id*

*tool_id*

*tool_inputs*

- Get the details on the "samtool sort" tool

```
detail_tool = gi.tools.show_tool(list_tool[0]['id'],io_details=True)
```

- Inputs information can be displayed with detail_tool['inputs'], use this to instantiate the inputs object

```
myinputs = inputs().set_dataset_param("input1", data_history['id'], src='hda')
                .set_param("sort_mode","")
```

- Now you can launch `samtool sort`

*history_id*

*tool_id*

*tool_inputs*

## Tools Object (3/3)

- Get the details on the "samtool sort" tool

```
detail_tool = gi.tools.show_tool(list_tool[0]['id'],io_details=True)
```

- Inputs information can be displayed with detail_tool['inputs'], use this to instantiate the inputs object

```
myinputs = inputs().set_dataset_param("input1", data_history['id'], src='hda')
                .set_param("sort_mode","")
```

- Now you can launch `samtool sort`

```
gi.tools.run_tool(new_history['id'], detail_tool['id'], myinputs)
```

return

## Workflows Object (1/2)

- We are going to use the same process now to Launch a Workflow
- To know what are the elements needed:

```
help(gi.workflows.run_workflow)
```

## Workflows Object (1/2)

- We are going to use the same process now to Launch a Workflow
- To know what are the elements needed:

```
help(gi.workflows.run_workflow)
```

```
run_workflow(self, workflow_id, dataset_map=None, params=None, history_id=Non
    history_name=None, import_inputs_to_history=False, replacement_params=Non
    method of bioblend.galaxy.workflows.WorkflowClient instance
    Run the workflow identified by ``workflow_id``

    :type workflow_id: string : Encoded workflow ID

    :type dataset_map: string or dict : A mapping of workflow inputs to datas
```

## Workflows Object (1/2)

- We are going to use the same process now to Launch a Workflow
- To know what are the elements needed:

```
help(gi.workflows.run_workflow)
```

```
run_workflow(self, workflow_id, dataset_map=None, params=None, history_id=Nor
    history_name=None, import_inputs_to_history=False, replacement_params=Nor
    method of bioblend.galaxy.workflows.WorkflowClient instance
    Run the workflow identified by ``workflow_id``

    :type workflow_id: string : Encoded workflow ID

    :type dataset_map: string or dict : A mapping of workflow inputs to datas
```

- Get the workflows list using the *get_workflows* method from the class Workflow
- Get the worflow details using the *show_workflow*

## Workflows Object (1/2)

- We are going to use the same process now to Launch a Workflow
- To know what are the elements needed:

```
help(gi.workflows.run_workflow)
```

```
run_workflow(self, workflow_id, dataset_map=None, params=None, history_id=Non
    history_name=None, import_inputs_to_history=False, replacement_params=Non
    method of bioblend.galaxy.workflows.WorkflowClient instance
    Run the workflow identified by ``workflow_id``

    :type workflow_id: string : Encoded workflow ID

    :type dataset_map: string or dict : A mapping of workflow inputs to datas
```

- Get the workflows list using the *get_workflows* method from the class Workflow
- Get the worflow details using the *show_workflow*

```
my_workflow = gi.workflows.get_workflows(name="wf_formation (imported from uploade
detailworkflow = gi.workflows.show_workflow(my_workflow[0]['id'])
```

## Workflows Object (2/2)

- Build the *dataset_map* (*key* = *"input workflow id key"* *value* = {*id* : *'dataset id'*, *src* : *'location of the data'*})

```
dataset_map = {}
dataset_map[detailworkflow['inputs'].keys()[0]] = {'id' : data_history['id'],
                                                    'src' : 'hda'}
```

- Launch the workflow

# Workflows Object (2/2)

- Build the *dataset_map* (*key* = *"input workflow id key"* *value* = {*id* : *'dataset id'*, *src* : *'location of the data'*})

```
dataset_map = {}
dataset_map[detailworkflow['inputs'].keys()[0]] = {'id' : data_history['id'],
                                                    'src' : 'hda'}
```

- Launch the workflow

```
gi.workflows.run_workflow(detailworkflow['id'], dataset_map=dataset_map,
                          history_id=new_history['id'])
```

## Workflows Object (2/2)

- Build the *dataset_map* (*key* = *"input workflow id key" value* = {*id : 'dataset id', src : 'location of the data'*})

```
dataset_map = {}
dataset_map[detailworkflow['inputs'].keys()[0]] = {'id' : data_history['id'],
                                                'src' : 'hda'}
```

- Launch the workflow

```
gi.workflows.run_workflow(detailworkflow['id'], dataset_map=dataset_map,
                     history_id=new_history['id'])
```

**Now you can try to test Bioblend with other tools and workflows.**

**Thank you for your attention**

return