INNUca

Release 1

Diogo N. Silva

Jan 20, 2018

Contents:

INNUca-NF is a NextFlow implementation of the INNUENDO pipeline designed for the high quality assembly, quality control and annotation of bacterial genomes. As a NextFlow pipeline, it can be easily deployed in many computing environments with minimal requirements.

CHAPTER 1

Requirements

- Java 8
- Docker
- NextFlow

1.1 generator package

1.1.1 Submodules

1.1.1.1 generator.HeaderSkeleton module

1.1.1.2 generator.Process module

class generator.Process.**Process**(*ptype*, *template*, *process_id=None*)

Bases: object

Main interface for basic process functionality

The Process class is intended to be inherited by specific process classes (e.g., *IntegrityCoverage*) and provides the basic functionality to build the channels and links between processes.

Child classes are expected to inherit the __init__ execution, which basically means that at least, the child must be defined as:

```
class ChildProcess(Process):
    def__init__(self, **kwargs):
        super().__init__(**kwargs)
```

This ensures that when the ChildProcess class is instantiated, it automatically sets the attributes of the parent class.

This also means that child processes must be instantiated providing information on the process type and jinja2 template with the nextflow code.

Parameters **ptype** : str

Process type. See *Process.accepted_types*.

template : str

Name of the jinja2 template with the nextflow code for that process. Templates are stored in `generator/templates`.

Attributes

template_str	Class property that returns a populated template string

Methods

render(template, context)	Wrapper to the jinja2 render method from a template file
set_channels(**kwargs)	General purpose method that sets the main channels
set_secondary_channel(source, channel_list)	General purpose method for setting a secondary channel

accepted_types = None
list: Accepted process types

pid = None
int: Process ID number that represents the order and position in the generated pipeline

process_id = None
int or str: optional Process ID that has no effect on the setup of the pipeline channels. It's used for the POST requests of each main process and is mapped to the process IDs of the innuendo/oneida platform

ptype = None
str: Process type. See *accepted_types*.

template = None
str: Template name for the current process. This string will be used to fetch the file containing the corresponding jinja2 template in the *_set_template()* method

_template_path = None
str: Path to the file containing the jinja2 template file. It's set in *_set_template()*.

input_type = None
str: Type of expected input data. Used to verify the connection between two processes is viable.

output_type = None
str: Type of output data. Used to verify the connection between two processes is viable.

ignore_type = None
boolean: If True, this process will ignore the input/output type requirements. This attribute is set to True for terminal singleton forks in the pipeline.

ignore_pid = None
boolean: If True, this process will not make the pid advance. This is used for terminal forks before the end of the pipeline.

dependencies = None
list: Contains the dependencies of the current process in the form of the *Process.template* attribute (e.g., [`fastqc`])

_main_in_str = None
str: String used to specify the prefix of main input channel.

_main_out_str = None
str: String used to specify the prefix of main output channel.

_input_channel = None
str: Place holder of the main input channel for the current process. This attribute can change dynamically depending on the forks and secondary channels in the final pipeline.

_output_channel = None
str: Place holder of the main output channel for the current process. This attribute can change dynamically depending on the forks and secondary channels in the final pipeline.

link_start = None
list: List of strings with the starting points for secondary channels. When building the pipeline, these strings will be matched with equal strings in the *link_end* attribute of other Processes.

link_end = None
list: List of dictionaries containing the a string of the ending point for a secondary channel. Each dictionary should contain at least two key/vals: `{"link": <link string>, "alias":<string for template>}`

status_channels = None
list: Name of the status channels produced by the process. By default, it sets a single status channel. If more than one status channels are required for the process, list each one in this attribute (e.g., *FastQC.status_channels*)

status_strs = None
str: Name of the status channel for the current process. These strings will be provided to the StatusCompiler process to collect and compile status reports

forks = None
list: List of strings with the literal definition of the forks for the current process, ready to be added to the template string.

_context = None
dict: Dictionary with the keyword placeholders for the string template of the current process.

_set_template(*template*)
Sets the path to the appropriate jinja template file

When a Process instance is initialized, this method will fetch the location of the appropriate template file, based on the `template` argument. It will raise an exception is the template file is not found. Otherwise, it will set the `Process.template_path` attribute.

_set_main_channel_name(*ptype*)
Sets the prefix for the main channel depending on the process type

`Pre-assembly` types are set to `MAIN_fq`, while `post-assembly` are set to `MAIN_assembly`. This distinction is important to allow the forking of the last main channel with FastQ files or with assembly files.

static render(*template*, *context*)
Wrapper to the jinja2 render method from a template file

Parameters **template** : str

Path to template file.

context : dict

Dictionary with kwargs context to populate the template

template_str
Class property that returns a populated template string

This property allows the template of a particular process to be dynamically generated and returned when doing `Process.template_str`.

Returns **x** : str

String with the complete and populated process template

set_channels(***kwargs*)
General purpose method that sets the main channels

This method will take a variable number of keyword arguments to set the *Process._context* attribute with the information on the main channels for the process. This is done by appending the process ID (*Process.pid*) attribute to the input, output and status channel prefix strings. In the output channel, the process ID is incremented by 1 to allow the connection with the channel in the next process.

The `**kwargs` system for setting the *Process._context* attribute also provides additional flexibility. In this way, individual processes can provide additional information not covered in this method, without changing it.

Parameters **kwargs** : dict

Dictionary with the keyword arguments for setting up the template context

set_secondary_channel(*source*, *channel_list*)
General purpose method for setting a secondary channel

This method allows a given source channel to be forked into one or more channels and sets those forks in the *Process.forks* attribute. Both the source and the channels in the `channel_list` argument must be the final channel strings, which means that this method should be called only after setting the main channels.

If the source is not a main channel, this will simply create a fork or set for every channel in the `channel_list` argument list:

```
SOURCE_CHANNEL_1.into{SINK_1;SINK_2}
```

If the source is a main channel, this will apply some changes to the output channel of the process, to avoid overlapping main output channels. For instance, forking the main output channel for process 2 would create a `MAIN_2.into{...}`. The issue here is that the `MAIN_2` channel is expected as the input of the next process, but now is being used to create the fork. To solve this issue, the output channel is modified into `_MAIN_2`, and the fork is set to the channels provided channels plus the `MAIN_2` channel:

```
_MAIN_2.into{MAIN_2;MAIN_5;...}
```

Parameters **source** : str

String with the name of the source channel

channel_list : list

List of channels that will receive a fork of the secondary channel

class generator.Process.**Status**(***kwargs*)
Bases: *generator.Process.Process*

Extends the Process methods to status-type processes

Attributes

template_str	Class property that returns a populated template string

Methods

render(template, context)	Wrapper to the jinja2 render method from a template file
set_channels(**kwargs)	General purpose method that sets the main channels
set_secondary_channel(source, channel_list)	General purpose method for setting a secondary channel
set_status_channels(channel_list)	General method for setting the input channels for the status process

set_status_channels(*channel_list*)

General method for setting the input channels for the status process

Given a list of status channels that are gathered during the pipeline construction, this method will automatically set the input channel for the status process. This makes use of the `mix` channel operator of nextflow for multiple channels:

```
STATUS_1.mix(STATUS_2,STATUS_3,...)
```

This will set the `status_channels` key for the `_context` attribute of the process.

Parameters **channel_list** : list

List of strings with the final name of the status channels

class generator.Process.**Init**(***kwargs*)

Bases: *generator.Process.Process*

Attributes

template_str	Class property that returns a populated template string

Methods

render(template, context)	Wrapper to the jinja2 render method from a template file
set_channels(**kwargs)	General purpose method that sets the main channels
set_secondary_channel(source, channel_list)	

set_secondary_channel(*source*, *channel_list*)

class generator.Process.**IntegrityCoverage**(***kwargs*)

Bases: *generator.Process.Process*

Process template interface for first integrity_coverage process

This process is set with:

- `input_type`: fastq
- `output_type`: fastq

- `ptype`: pre_assembly

It contains two **secondary channel link starts**:

- `SIDE_phred`: Phred score of the FastQ files
- `SIDE_max_len`: Maximum read length

Attributes

`template_str`	Class property that returns a populated template string

Methods

`render`(template, context)	Wrapper to the jinja2 render method from a template file
`set_channels`(**kwargs)	General purpose method that sets the main channels
`set_secondary_channel`(source, channel_list)	General purpose method for setting a secondary channel

class `generator.Process.`**`SeqTyping`**(***kwargs*)

Bases: *generator.Process.Process*

Attributes

`template_str`	Class property that returns a populated template string

Methods

`render`(template, context)	Wrapper to the jinja2 render method from a template file
`set_channels`(**kwargs)	General purpose method that sets the main channels
`set_secondary_channel`(source, channel_list)	General purpose method for setting a secondary channel

class `generator.Process.`**`PathoTyping`**(***kwargs*)

Bases: *generator.Process.Process*

Attributes

`template_str`	Class property that returns a populated template string

Methods

`render`(template, context)	Wrapper to the jinja2 render method from a template file
`set_channels`(**kwargs)	General purpose method that sets the main channels
`set_secondary_channel`(source, channel_list)	General purpose method for setting a secondary channel

class generator.Process.**CheckCoverage**(***kwargs*)
Bases: *generator.Process.Process*

Process template interface for additional integrity_coverage process

This process is set with:

- input_type: fastq
- output_type: fastq
- ptype: pre_assembly

It contains one **secondary channel link start**:

- SIDE_max_len: Maximum read length

Attributes

template_str	Class property that returns a populated template string

Methods

render(template, context)	Wrapper to the jinja2 render method from a template file
set_channels(**kwargs)	General purpose method that sets the main channels
set_secondary_channel(source, channel_list)	General purpose method for setting a secondary channel

class generator.Process.**FastQC**(***kwargs*)
Bases: *generator.Process.Process*

FastQC process template interface

This process is set with:

- input_type: fastq
- output_type: fastq
- ptype: pre_assembly

It contains two **status channels**:

- STATUS_fastqc: Status for the fastqc process
- STATUS_report: Status for the fastqc_report process

Attributes

template_str	Class property that returns a populated template string

Methods

render(template, context)	Wrapper to the jinja2 render method from a template file
set_channels(**kwargs)	General purpose method that sets the main channels

Continued on next page

Table 1.16 – continued from previous page

set_secondary_channel(source, channel_list)	General purpose method for setting a secondary channel

status_channels = None
list: Setting status channels for FastQC execution and FastQC report

class generator.Process.**Trimmomatic**(**kwargs)
Bases: *generator.Process.Process*

Trimmomatic process template interface

This process is set with:

- input_type: fastq
- output_type: fastq
- ptype: pre_assembly

It contains one **secondary channel link end**:

- SIDE_phred (alias: SIDE_phred): Receives FastQ phred score

Attributes

template_str	Class property that returns a populated template string

Methods

render(template, context)	Wrapper to the jinja2 render method from a template file
set_channels(**kwargs)	General purpose method that sets the main channels
set_secondary_channel(source, channel_list)	General purpose method for setting a secondary channel

class generator.Process.**FastqcTrimmomatic**(**kwargs)
Bases: *generator.Process.Process*

Fastqc + Trimmomatic process template interface

This process executes FastQC only to inform the trim range for trimmomatic, not for QC checks.

This process is set with:

- input_type: fastq
- output_type: fastq
- ptype: pre_assembly

It contains one **secondary channel link end**:

- SIDE_phred (alias: SIDE_phred): Receives FastQ phred score

It contains three **status channels**:

- STATUS_fastqc: Status for the fastqc process
- STATUS_report: Status for the fastqc_report process
- STATUS_trim: Status for the trimmomatic process

Attributes

template_str	Class property that returns a populated template string

Methods

render(template, context)	Wrapper to the jinja2 render method from a template file
set_channels(**kwargs)	General purpose method that sets the main channels
set_secondary_channel(source, channel_list)	General purpose method for setting a secondary channel

class generator.Process.**Spades**(***kwargs*)

Bases: *generator.Process.Process*

Spades process template interface

This process is set with:

- input_type: fastq
- output_type: assembly
- ptype: assembly

It contains one **secondary channel link end**:

- SIDE_max_len (alias: SIDE_max_len): Receives max read length

Attributes

template_str	Class property that returns a populated template string

Methods

render(template, context)	Wrapper to the jinja2 render method from a template file
set_channels(**kwargs)	General purpose method that sets the main channels
set_secondary_channel(source, channel_list)	General purpose method for setting a secondary channel

class generator.Process.**ProcessSpades**(***kwargs*)

Bases: *generator.Process.Process*

Process spades process template interface

This process is set with:

- input_type: assembly
- output_type: assembly
- ptype: post_assembly

Attributes

template_str	Class property that returns a populated template string

Methods

render(template, context)	Wrapper to the jinja2 render method from a template file
set_channels(**kwargs)	General purpose method that sets the main channels
set_secondary_channel(source, channel_list)	General purpose method for setting a secondary channel

class generator.Process.**AssemblyMapping**(***kwargs*)

Bases: *generator.Process.Process*

Assembly mapping process template interface

This process is set with:

- input_type: assembly
- output_type: assembly
- ptype: post_assembly

It contains one **secondary channel link end**:

- MAIN_fq (alias: _MAIN_assembly): Receives the FastQ files

  from the last process with fastq output type.

It contains two **status channels**:

- STATUS_am: Status for the assembly_mapping process
- STATUS_amp: Status for the process_assembly_mapping process

Attributes

template_str	Class property that returns a populated template string

Methods

render(template, context)	Wrapper to the jinja2 render method from a template file
set_channels(**kwargs)	General purpose method that sets the main channels
set_secondary_channel(source, channel_list)	General purpose method for setting a secondary channel

class generator.Process.**Pilon**(***kwargs*)

Bases: *generator.Process.Process*

Pilon mapping process template interface

This process is set with:

- input_type: assembly
- output_type: assembly
- ptype: post_assembly

It contains one **dependency process**:

- `assembly_mapping`: Requires the BAM file generated by the

assembly mapping process

Attributes

`template_str`	Class property that returns a populated template string

Methods

`render`(template, context)	Wrapper to the jinja2 render method from a template file
`set_channels`(**kwargs)	General purpose method that sets the main channels
`set_secondary_channel`(source, channel_list)	General purpose method for setting a secondary channel

class generator.Process.**Mlst**(***kwargs*)

Bases: *generator.Process.Process*

Mlst mapping process template interface

This process is set with:

- `input_type`: assembly
- `output_type`: None
- `ptype`: post_assembly

It contains one **secondary channel link end**:

- `MAIN_assembly` (alias: `MAIN_assembly`): Receives the last

assembly.

Attributes

`template_str`	Class property that returns a populated template string

Methods

`render`(template, context)	Wrapper to the jinja2 render method from a template file
`set_channels`(**kwargs)	General purpose method that sets the main channels
`set_secondary_channel`(source, channel_list)	General purpose method for setting a secondary channel

class generator.Process.**Abricate**(***kwargs*)

Bases: *generator.Process.Process*

Abricate mapping process template interface

This process is set with:

- `input_type`: assembly

- output_type: None
- ptype: post_assembly

It contains one **secondary channel link end**:

- MAIN_assembly (alias: MAIN_assembly): Receives the last assembly.

Attributes

template_str	Class property that returns a populated template string

Methods

render(template, context)	Wrapper to the jinja2 render method from a template file
set_channels(**kwargs)	General purpose method that sets the main channels
set_secondary_channel(source, channel_list)	General purpose method for setting a secondary channel

class generator.Process.**Prokka**(***kwargs*)

Bases: *generator.Process.Process*

Prokka mapping process template interface

This process is set with:

- input_type: assembly
- output_type: None
- ptype: post_assembly

It contains one **secondary channel link end**:

- MAIN_assembly (alias: MAIN_assembly): Receives the last assembly.

Attributes

template_str	Class property that returns a populated template string

Methods

render(template, context)	Wrapper to the jinja2 render method from a template file
set_channels(**kwargs)	General purpose method that sets the main channels
set_secondary_channel(source, channel_list)	General purpose method for setting a secondary channel

class generator.Process.**Chewbbaca**(***kwargs*)

Bases: *generator.Process.Process*

Prokka mapping process template interface

This process is set with:

- `input_type`: assembly
- `output_type`: None
- `ptype`: post_assembly

It contains one **secondary channel link end**:

- `MAIN_assembly` (alias: `MAIN_assembly`): Receives the last assembly.

Attributes

`template_str`	Class property that returns a populated template string

Methods

`render`(template, context)	Wrapper to the jinja2 render method from a template file
`set_channels`(**kwargs)	General purpose method that sets the main channels
`set_secondary_channel`(source, channel_list)	General purpose method for setting a secondary channel

class generator.Process.**TraceCompiler**(***kwargs*)

Bases: *generator.Process.Process*

Attributes

`template_str`	Class property that returns a populated template string

Methods

`render`(template, context)	Wrapper to the jinja2 render method from a template file
`set_channels`(**kwargs)	General purpose method that sets the main channels
`set_secondary_channel`(source, channel_list)	General purpose method for setting a secondary channel

class generator.Process.**StatusCompiler**(***kwargs*)

Bases: *generator.Process.Status*

Status compiler process template interface

This special process receives the status channels from all processes in the generated pipeline.

Attributes

`template_str`	Class property that returns a populated template string

Methods

render(template, context)	Wrapper to the jinja2 render method from a template file
set_channels(**kwargs)	General purpose method that sets the main channels
set_secondary_channel(source, channel_list)	General purpose method for setting a secondary channel
set_status_channels(channel_list)	General method for setting the input channels for the status process

1.1.2 Module contents

1.2 nextflow_generator module

exception nextflow_generator.**ProcessError**(*value*)
Bases: Exception

exception nextflow_generator.**ChannelError**(*p1*, *p2*, *t1*, *t2*)
Bases: Exception

class nextflow_generator.**NextflowGenerator**(*process_list*, *nextflow_file*, *process_ids=None*)
Bases: object

Methods

build()	Main pipeline builder

process_map = {'assembly_mapping': <class 'generator.Process.AssemblyMapping'>, 'prok
dict: Maps the process ids to the corresponding template interface class

processes = None
list: Stores the process interfaces in the specified order

nf_file = None
str: Path to file where the pipeline will be generated

template = None
str: String that will harbour the pipeline code

secondary_channels = None
dict: Stores secondary channel links

status_channels = None
list: Stores the status channels from each process

_check_pipeline_requirements()
Checks for some pipeline requirements before building

Currently, the only hard requirement is that the pipeline must start with the integrity_coverage process, in order to evaluate if the input FastQ are corrupt or not.

Besides this requirements, it checks for the existence the dependencies for all processes.

_build_header()
Adds the header template to the master template string

_set_channels()
Sets the main channels for the pipeline

The setup of the main channels follows four main steps for each process specified in the *NextflowGenerator.processes* attribute:

- (If not the first process) Checks if the input of the current

process is compatible with the output of the previous process. - Checks if the current process has starts any secondary channels. If so, populate the *NextflowGenerator.secondary_channels* with the name of the link start, the process class and a list to harbour potential receiving ends. - Checks if the current process receives from any secondary channels. If a corresponding secondary link has been previously set, it will populate the *NextflowGenerator.secondary_channels* attribute with the receiving channels. - Sets the main channels by providing the process ID.

Notes

On the secondary channel setup: With this approach, there can only be one secondary link start for each type of secondary link. For instance, If there are two processes that start a secondary channel for the SIDE_max_len channel, only the last one will be recorded, and all receiving processes will get the channel from the latest process.

_set_secondary_channels()
Sets the secondary channels for the pipeline

This will iterate over the *NextflowGenerator.secondary_channels* dictionary that is populated when executing *NextflowGenerator._set_channels()* method.

_set_status_channels()
Compiles all status channels for the status compiler process

build()
Main pipeline builder

This method is responsible for building the *NextflowGenerator.template* attribute that will contain the nextflow code of the pipeline.

First it builds the header, then sets the main channels, the secondary channels and finally the status channels. When the pipeline is built, is writes the code to a nextflow file.

nextflow_generator.**get_args**()

nextflow_generator.**get_tuples**(*task*)

nextflow_generator.**copy_project**(*path*)

Parameters path

nextflow_generator.**main**(*args*)

1.3 templates package

1.3.1 Submodules

1.3.1.1 assembly_report module

1.3.1.2 fastqc module

1.3.1.3 fastqc_report module

1.3.1.4 integrity_coverage module

1.3.1.5 process_abricate module

1.3.1.6 process_assembly_mapping module

1.3.1.7 process_spades module

1.3.1.8 spades module

1.3.1.9 trimmomatic module

1.3.1.10 trimmomatic_report module

1.3.2 Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

g

n

Index

Symbols

A

B

C

D

F

G

I

L

M

N

O

P

R

S

T