

---

# **BLUEPRINT Pipeline Documentation**

***Release 0.1***

**Johannes Alneberg, Anders Andersson**

November 23, 2016



<b>1</b>	<b>Purpose</b>	<b>3</b>
<b>2</b>	<b>Contents:</b>	<b>5</b>
2.1	Usage . . . . .	5
2.2	Implementation . . . . .	10



This documentation aim to cover the purpose and usage of a bioinformatics toolbox developed for the Blueprint project. Blueprint is a BONUS project and you can read more about it at the [Blueprint webpage](#) and about BONUS [here](#).



---

### Purpose

---

This toolbox or pipeline's aim is to be used to process metagenomic and metatranscriptomic sequencing data, produced by the Blueprint project, in order to present the results in a meaningful and comprehensive way. The basis for the pipeline will be a reference metagenomic assembly, that will only be generated a few times. Both metagenomic and metatranscriptomic samples will be mapped against the reference assembly to quantify the presence of each reference contig or reference open reading frame in the sample. When a new such assembly is performed all results derived from the assembly needs to be recomputed.

A big advantage in using a reference metagenomic assembly is that several computationally heavy steps, e.g. functional annotation and taxonomic classification, can be done once for the reference assembly and then reused for all samples. As an example the taxonomic classification will be done per sequence in the reference assembly, and to estimate the abundance of a specific taxa in a sample, the pipeline looks at the all sequences classified as this taxa in the mapping of the sample sequences against the reference assembly.

The outputs of the pipeline are files containing a quantitative measure for each gene or open reading frame in the reference assembly. Together with the annotation for the reference assembly, this is sufficient to quantify different annotations for each sample.

As a quantification measure, RPKM, reads per kilobase per million reads, have been chosen since this normalizes the number of reads mapping to a gene with the length of the gene and the total number of reads in the sequencing output.





---

**Contents:**

---

## 2.1 Usage

This section describes how to install and use the pipeline. It is written assuming you are using a linux computer but the same instructions should be valid on a Mac OSX computer as well. The pipeline uses external software that needs to be installed before attempting to run the pipeline. The software that needs to be available is:

- [Bowtie2](#)
- [samtools](#)
- [sickle](#)
- [Fastqc](#)

### 2.1.1 Setting up the python environment

Unfortunately this pipeline needs to have two different python environments available, since it needs to have two different versions (2.7 and 3.4) of python available. This instructions will setup these two environments for you using miniconda from [Continuum](#). If you're already using miniconda or anaconda, you might want to edit the commands to suit your needs.

First download and install miniconda in your home directory:

```
cd ~
wget http://repo.continuum.io/miniconda/Miniconda-latest-Linux-x86_64.sh -O miniconda.sh
chmod u+x miniconda.sh
./miniconda.sh -b
```

Now we are ready to create the two environments.:

```
conda create -n BLUEPRINT_pipeline python=3.4
conda create -n BLUEPRINT_pipeline_2.7 python=2.7

source activate BLUEPRINT_pipeline_2.7
conda install --yes pip numpy scipy pandas
pip install pysam
source deactivate

source activate BLUEPRINT_pipeline
conda install pip snakemake
source deactivate
```

This will create one environment named `BLUEPRINT_pipeline` which uses python version 3.4 and will be the main version of python used for the pipeline. The other environment `BLUEPRINT_pipeline2.7` is used for a custom script that is used to quantify the open reading frames used as a part of the pipeline.

We're now ready to install the pipeline itself, starting off by downloading it from github:

```
wget https://github.com/EnvGen/BLUEPRINT_pipeline/archive/master.zip
unzip master.zip

cd BLUEPRINT_pipeline-master
```

If these commands are successful, you should now have everything ready to run the small test example below.

### 2.1.2 Running an example

With the pipeline source code, a small test data example is supplied. By running this example, we aim to show how the pipeline works in practice. First, let's move into the test directory:

```
cd test
```

This directory currently contains a few files, the output of the excellent 'tree' command shows us:

```
$ tree
.
-- config.json
-- config_uppmax.json
-- Snakefile
-- test_data
    -- annotation
    |   -- reference
    |       -- assembly_v1.gff
    -- references
    |   -- assembly_v1.fna
    -- samples
    -- after_gc
    |   -- 120322_R1.fastq
    |   -- 120322_R2.fastq
    |   -- 120507_R1.fastq
    |   -- 120507_R2.fastq
    -- raw
    -- 120322_R1.fastq
    -- 120322_R2.fastq
    -- 120507_R1.fastq
    -- 120507_R2.fastq

7 directories, 13 files
```

The files within the `test_data` directory is the data that we'll use to kick off our pipeline, the Snakefile defines what result files we'd like to create and how to create them, the `config.json` file defines specific configurations we'll need (for this case where the python2 executable is available), and `config_uppmax.json` is a special config file needed only if we're running our test on any of the `uppmax` clusters.

We'll use the command `snakemake` to run our pipeline. The first command will execute a rule in the Snakefile that creates a directory structure that will suite the pipeline, and creates links to the `test_data` files needed:

```
snakemake prepare
```

Lets have a look at what this command created:

```
$ tree
.
-- annotation
|   -- reference
|       -- assembly_v1.gff
-- config.json
-- config_uppmax.json
-- mapping
-- quantification
-- references
|   -- assembly_v1.fna
-- rpkm_for_orfs.py -> ~/repos/BLEUPRINT_pipeline/test/./scripts/rpkm_for_orfs.py
-- samples
|   -- after_qc
|       |   -- 120322_R1.fastq
|       |   -- 120322_R2.fastq
|       |   -- 120507_R1.fastq
|       |   -- 120507_R2.fastq
|   -- raw
|       -- 120322_R1.fastq
|       -- 120322_R2.fastq
|       -- 120507_R1.fastq
|       -- 120507_R2.fastq
-- Snakefile
-- test_data
    -- annotation
    |   -- reference
    |       -- assembly_v1.gff
    -- references
    |   -- assembly_v1.fna
    -- samples
        -- after_qc
        |   -- 120322_R1.fastq
        |   -- 120322_R2.fastq
        |   -- 120507_R1.fastq
        |   -- 120507_R2.fastq
        -- raw
        -- 120322_R1.fastq
        -- 120322_R2.fastq
        -- 120507_R1.fastq
        -- 120507_R2.fastq

15 directories, 24 files
```

This shows us that the command has created a directory structure similar to the one present in the 'test\_data' directory and copied the files present in test\_data. It has also created two new directories named mapping and quantification where some output from the pipeline will be stored and created a link to the script *rpkm\_for\_orfs.py*. Now we should check what the pipeline would do if we executed it. By adding the *--dryrun* argument to *snakemake*, it will not execute any command but only show what it would do:

```
snakemake --dryrun test_qc
```

This should output a list of rules with input files and output files connected to them. After going through this list, running the first part of the pipeline should now be as simple as:

```
snakemake test_qc
```

If everything went alright you should now have the following files:

```
$ tree
.
-- annotation
|   -- reference
|       -- assembly_v1.gff
-- config.json
-- config_uppmax.json
-- mapping
-- quantification
-- references
|   -- assembly_v1.fna
-- rpkm_for_orfs.py -> /pica/h1/alneberg/repos/BLEPRINT_pipeline/test/./scripts/rpkm_for_orfs.py
-- samples
|   -- after_gc
|       |   -- 120322_R1.fastq
|       |   -- 120322_R2.fastq
|       |   -- 120507_R1.fastq
|       |   -- 120507_R2.fastq
|   -- fastqc
|       |   -- 120322
|       |       |   -- 120322_R1_fastqc.html
|       |       |   -- 120322_R2_fastqc.html
|       |   -- 120507
|       |       |   -- 120507_R1_fastqc.html
|       |       |   -- 120507_R2_fastqc.html
|   -- raw
|       |   -- 120322_R1.fastq
|       |   -- 120322_R2.fastq
|       |   -- 120507_R1.fastq
|       |   -- 120507_R2.fastq
|   -- sickle
|       |   -- 120322.log
|       |   -- 120322_R1.fastq
|       |   -- 120322_R2.fastq
|       |   -- 120322_single.fastq
|       |   -- 120507.log
|       |   -- 120507_R1.fastq
|       |   -- 120507_R2.fastq
|       |   -- 120507_single.fastq
-- Snakefile
-- test_data
    -- annotation
    |   -- reference
    |       -- assembly_v1.gff
    -- references
    |   -- assembly_v1.fna
    -- samples
        -- after_gc
        |   -- 120322_R1.fastq
        |   -- 120322_R2.fastq
        |   -- 120507_R1.fastq
        |   -- 120507_R2.fastq
        -- raw
        |   -- 120322_R1.fastq
        |   -- 120322_R2.fastq
        |   -- 120507_R1.fastq
        |   -- 120507_R2.fastq
```

```
19 directories, 36 files
```

At this point, in a real case, you should have a look at the fastqc output files with a *.html* extension. These are reports about the quality of the input files and based on these the user will have to take a decision if the input files are good enough to continue with, in that case copy the files to the *after\_qc* directory, or if some additional step has to be run, such as cutting adaptor sequences. For this example we've prepared this step already, so we're ready to take the next step. To check what the next step will execute, check:

```
snakemake --dryrun all_from_mapping
```

and to kick off the last main part of the pipeline, run:

```
snakemake all_from_mapping
```

If everything went alright you should now have have the following files:

```
$tree
.
-- annotation
|   -- reference
|       -- assembly_v1.gff
-- config.json
-- config_uppmax.json
-- mapping
|   -- bowtie2
|       -- assembly_v1
|           |   -- 120322
|               -- assembly_v1.1.bt2
|               -- assembly_v1.2.bt2
|               -- assembly_v1.3.bt2
|               -- assembly_v1.4.bt2
|               -- assembly_v1.rev.1.bt2
|               -- assembly_v1.rev.2.bt2
-- quantification
|   -- assembly_v1
|       -- orf
|           -- 120322
|               |   -- 120322.rpkm
|               -- 120507
|                   -- 120507.rpkm
-- references
|   -- assembly_v1.fna
-- rpkm_for_orfs.py -> ~/repos/BUEPRINT_pipeline/test/../../scripts/rpkm_for_orfs.py
-- samples
|   -- after_qc
|       |   -- 120322_R1.fastq
|       |   -- 120322_R2.fastq
|       |   -- 120507_R1.fastq
|       |   -- 120507_R2.fastq
|   -- fastqc
|       |   -- 120322
|       |       |   -- 120322_R1_fastqc.html
|       |       |   -- 120322_R2_fastqc.html
|       |   -- 120507
|       |       -- 120507_R1_fastqc.html
|       |       -- 120507_R2_fastqc.html
|   -- raw
|       |   -- 120322_R1.fastq
|       |   -- 120322_R2.fastq
```

```
| | -- 120507_R1.fastq
| | -- 120507_R2.fastq
| -- sickle
|   -- 120322.log
|   -- 120322_R1.fastq
|   -- 120322_R2.fastq
|   -- 120322_single.fastq
|   -- 120507.log
|   -- 120507_R1.fastq
|   -- 120507_R2.fastq
|   -- 120507_single.fastq
-- Snakefile
-- test_data
  -- annotation
  | -- reference
  |   -- assembly_v1.gff
-- references
  | -- assembly_v1.fna
-- samples
  -- after_qc
  | -- 120322_R1.fastq
  | -- 120322_R2.fastq
  | -- 120507_R1.fastq
  | -- 120507_R2.fastq
  -- raw
  -- 120322_R1.fastq
  -- 120322_R2.fastq
  -- 120507_R1.fastq
  -- 120507_R2.fastq

27 directories, 50 files
```

Where the two files *120322.rpkm* and *120507.rpkm* are the most interesting ones. These should contain one row for each open reading frame found in the file *annotation/reference/assembly\_v1.gff*. Each row would then contain the ORF id and a RPKM value which is ready to e.g. be imported into a database. If you'd like to remove all the files created in this exercise in a smooth way there is a special rule for that as well:

```
# Running this will delete all directories created by the prepare command
snakemake clean_up
```

Now you're ready to start all over again with the *snakemake prepare* command.

## 2.2 Implementation

The toolbox consists of a set of rules implemented in [Snakemake](#). These are all found under their specific category within the 'rules' directory.