# cpm_pypeline Documentation

***Release 0.0.1***

**Francesco Favero**

**Jul 23, 2018**

# Contents

A simple and lightweight python framework to organize bioinformatics analyses.

# CHAPTER 1

## About:

Bioinformatics tasks require a long list of parameters: genome files, indexes, database, results of previous analyses and many more. It is not simple to keep records of version of softwares and databases and guarantee the reproducibility of an analysis.

This framework facilitate the organization of programs and files (eg, genomes and databases) in structured YAML files, defined in the framework interface as *profiles*. In addition, it provides a way to organize and execute bioinformatic tasks: each base task is wrapped in a python script, called *snippets*. A *snippet* take advantage of the *profile*, as a result, the number of arguments required to execute the script are greatly diminished. A number of *snippets* can be chained in a YAML file, to form a *pipeline*. A *pipeline* is a series of tasks that need to be executed in a given order, by a queuing system (eg, serial execution, parallel execution, torque/moab...). *Snippets* and *pipelines* take advantage of the `argparse` python library to provide command line documentation upon execution.

CHAPTER 2

# Quick Intro

## 2.1 In a nutshell:

### 2.1.1 Available presets

There are available preset of snippets and pipelines. You can access the list of repository and manage the installed modules with the *repos* subcommand

```
$ pype repos
usage: pype repos [-r REPO_LIST] {list,install,clean,info} ...

positional arguments:
  {list,install,clean,info}
  list                  List the available repositories
  install               Install modules from selected repository
  clean                 Cleanup all module folders
  info                  Print location of the modules currently in use

optional arguments:
  -r REPO_LIST, --repo REPO_LIST
                        Repository list. Default:
                        /usr/lib/python2.7/site-packages/pype/repos.yaml
```

### 2.1.2 Overview of the framework

The framework is heavily based on the python `argparse` module aiming for a self explanatory use of the main command line interface: **pype**.

The management of tools version and sources (assembly/organisms) is organized in *profiles* files.

```
 $ pype profiles
 usage: pype profiles {info,check}
```

(continues on next page)

(continued from previous page)

```
 positional arguments:
 {info,check}
 info        Retrieve details from available profiles
 check       Check if a profile is valid

$ pype profiles info --all
     default:          Default profile b37 from 1k genome project
     hg19:             Profile for the UCSC hg19 assembly
     hg38:             Profile for the hg38 assembly human genome
```

Each profile is defined by a YAML file, an example:

```
info:
    description: HG19 default profile
    date:        10/03/2016
genome_build: 'hg19'
 files:
    genome_fa:    /path/to/b37/genome.fa
    genome_fa_gz: /path/to/b37//genome.fa.gz
    exons_list:   /path/to/exome_hg37_interval_list.txt
    db_snp:       /path/to/dbSNP/dbsnp_138.b37.vcf.gz
    cosmic:       /path/to/cosmic/CosmicCodingMuts.vcf.gz

 programs:
    samtools_0:
        path: samtools
        version: 0.1.18
    samtools_1:
        path: samtools
        version: 1.2
```

Each tools is wrapped into a python function (*snippets*), which is designed to load program specified in the selected profile, and usually execute the tools using the `subprocess` module.

```
$ pype snippets
error: too few arguments
usage: pype snippets [--log LOG]
                {biobambam_merge,mutect,bwa_mem,freebayes}
                ...

positional arguments:
{biobambam_merge,mutect,bwa_mem,freebayes}
biobambam_merge     Mark duplicates and merge BAMs with biobambam
mutect              Call germline and somatic variants with mutect
oncotator           Annotate various file format with Oncotator
bwa_mem             Align fastQ file with the BWA mem algorithm
freebayes           A haplotype-based variant detector
```

The tools can be chained and combined one-another within a YAML files (*pipelines*), which specify dependencies and input files. The *profile* is parsed to provide command line option by the *argparse* interface

```
$ pype pipelines
error: too few arguments
usage: pype pipelines [--queue {msub,echo,none}] [--log LOG]
                  {bwa,freebayes,mutect,bwa_mem}
```

(continues on next page)

(continued from previous page)

```
                ...

positional arguments:
{bwa,delly,freebayes,mutect,bwa_mem}
bwa                 Align with bwa mem, merge different lanes and
                    performs various QC stats
freebayes           Freebayes pipeline from Fastq to VCF
mutect              Mutect from fastq files
bwa_mem             Align with bwa mem and performs QC stats

optional arguments:
--queue {msub,echo,none}
                   Select the queuing system to run the pipeline
--log LOG          Path used to write the pipeline logs. Default
                   working directory.
```

A typical scenario that demonstrate the advantages offered by this package is the alignment of two DNA sequencing (NGS) samples, a normal and a tumor samples, and perform a comparison between two samples to find somatic mutations. In the command line this task require for each sample a command for the alignment, a command for sorting and a command for indexing. After the two samples are processed the results need to be compared by a *mutation caller*, usually specifying some database such as COSMIC or dbSNP.

The easiest and fastest way to implement the tasks is by wrapping the command line into bash script and submit them to a queuing system, or run it directly in a shell. This would impose a self organization of logs and scripts, and still possibli results in a lot of maintenance to keep the scripts up to date.

Other pipeline systems can solve the problem, but they may not be executable on the fly by command line interface (eg, need to configure file previous execution) or they may not allows to change genome build or database version easily.

with the *bio_pype* framework each step can be launched separately, example of usage of the alignment snippet:

```
pype snippets bwa_mem
usage:
pype snippets bwa_mem -h HEADER -1 F1 -2 F2 [-t TMP] [-o OUT]

optional arguments:
  -h HEADER          Header file containing the @RG groups,
                     or the comma separated header line
  -1 F1              First mate fastQ file
  -2 F2              Second mate fastQ file
  -t TMP, --tmp TMP  Temporary folder
  -o OUT, --out OUT  Output name for the bam file
```

or use the pipeline:

```
pype pipelines mutect
usage: pype pipelines mutect --tumor_bam TUMOR_BAM --out_dir OUT_DIR
                             --sample_name SAMPLE_NAME --normal_bam NORMAL_BAM
                             [--tmp_dir TMP_DIR] --tumor_bwa_list
                             TUMOR_BWA_LIST --qc_bam_tumor QC_BAM_TUMOR
                             --normal_bwa_list NORMAL_BWA_LIST --qc_bam_normal
                             QC_BAM_NORMAL

Required:
  Required pipeline arguments
```

(continues on next page)

(continued from previous page)

```
  --tumor_bam TUMOR_BAM
                        Input Bam file of the tumor sample, type: str
  --out_dir OUT_DIR     Output file name prefix for the analysis, type: str
  --sample_name SAMPLE_NAME
                        Sample name or identifier of the run, type: str
  --normal_bam NORMAL_BAM
                        Input Bam file of the normal/control sample, type: str
  --tumor_bwa_list TUMOR_BWA_LIST
                        Batch file to run the bwa_mem pipeline on different
                        lanes of the tumor, type: str
  --qc_bam_tumor QC_BAM_TUMOR
                        Tumor BAM files QC directory output path, type: str
  --normal_bwa_list NORMAL_BWA_LIST
                        Batch file to run the bwa_mem pipeline on different
                        lanes of the normal/control, type: str
  --qc_bam_normal QC_BAM_NORMAL
                        Normal/Control BAM files QC directory output path,
                        type: str

Optional:
  Optional pipeline arguments

  --tmp_dir TMP_DIR     temporary folder, type: str. Default: /scatch
```

### 2.1.3 Python and Environment Modules:

example of loading the hg37 profile in the python prompt:

```
from pype.modules.profiles import get_profiles

profiles = get_profiles({})
profile = profiles['hg37']
genome = profile.files['genome_fa_gz']
```

The framework also supports Environment Modules:

```
from pype.modules.profiles import get_profiles
from pype.env_modules import get_module_cmd, program_string

profiles = get_profiles({})
module('add', program_string(profile.programs['samtools_1']))
```

CHAPTER 3

# Installation

## 3.1 Installation from git:

**Note:** It is strongly advised to use virtualenv to install the module locally.

```
git clone https://bitbucket.org/ffavero/bio_pype
cd bio_pype
python setup.py test
python setup.py install
```

## 3.2 Using virtualenv:

This code guide thought the installation and use of virtualenv when the main python installation does not include the virtualenv module.

```
mkdir ~/venv_files
# Download virtualenv files (not installed in the python version
# used in computerome)
curl -L -o ~/venv_files/setuptools-20.2.2-py2.py3-none-any.whl \
   https://github.com/pypa/virtualenv/raw/develop/virtualenv_support/setuptools-20.2.
↪2-py2.py3-none-any.whl
curl -L -o ~/venv_files/pip-8.1.0-py2.py3-none-any.whl \
   https://github.com/pypa/virtualenv/raw/develop/virtualenv_support/pip-8.1.0-py2.
↪py3-none-any.whl
curl -L -o ~/venv_files/virtualenv.py \
   https://github.com/pypa/virtualenv/raw/develop/virtualenv.py

# Create a virtual environment
python ~/venv_files/virtualenv.py --extra-search-dir=~/venv_files \
```

(continues on next page)

(continued from previous page)

```
   ~/venv_bio_pype
# Activate the environment
source ~/venv_bio_pype/bin/activate

# Install bio_pype as instructed above from git

# The pype command line utility will be available upon the "activation"
# of the environment or by specifying the full path, in this case:

# ~/venv_bio_pype/bin/pype

# To deactivate the virtual env:
deactivate
```

CHAPTER 4

# Usage

## 4.1 Basic usage

Access first level modules:

```
$ pype
error: too few arguments
usage: pype [-p PROFILE] {snippets,profiles} ...

Slim and simple framework to ease the managements of data, tools and
pipelines in the computerome HPC

positional arguments:
    snippets        Snippets
    profiles        List of profiles

optional arguments:
  -p PROFILE, --profile PROFILE
                    Select profile to set reference genome and tools
                    versions

This is version 0.0.1 - Francesco Favero - 7 March 2016
```

Access submodules in the selected level:

```
$ pype snippets
error: too few arguments
usage: pype snippets {test} ...

positional arguments:
  {test}
    test  test_snippets
```

Execute or access submodule:

```
$ pype snippets test
error: argument --test is required
usage: pype snippets test --test TEST [-o OPT]

optional arguments:
  --test TEST  Test metavar
  -o OPT       test option
```

```
$ pype snippets test --test World
Hello World
```

```
$ pype snippets test --test mate -o Cheers
Cheers mate
```

It is possible to pass local folders where to pick custom snippets and profiles via environment variables:

```
$ PYPE_SNIPPETS="/path/to/custom_snippets" \
  PYPE_PROFILES="/path/to/custom_profiles" pype
```

Or alternatively it is possible to set custom **PYPE_SNIPPETS** and **PYPE_PROFILES** in the *~/.bashrc* (or alternatives) file.

CHAPTER 5

# API reference

## 5.1 Snippets

### 5.1.1 Writing new Snippets:

The snippets are located in a python module (mind the __init__.py in the folder containing the snippets). In order to function, each snippet need to have 4 specific function:

1. *requirements*: a function returning a dictionary with the necessary resource to run the snippet (used to allocate resource in queuing systems)
2. *results*: a function accepting a dictionary with the snippet arguments and returning a dictionary listing all the files produced by the execution of the snippet
3. *add_parser*: a function that implement the `argparse` module and defines the command line arguments accepted by the snippet
4. a function named as the snippet file name (without the *.py* extension), containing the code for the execution of the tool

A very minimalistic snippets structure

```
def requirements():
    return({'ncpu': 1, 'time': '00:01:00'})


def results(argv):
    return([])


def add_parser(subparsers, module_name):
    return subparsers.add_parser(module_name, help='test_snippets', add_help=False)
```

(continues on next page)

(continued from previous page)

```
def test_args(parser, subparsers, argv):
    parser.add_argument('--test', dest='test',
                        help='Test metavar', required=True)
    parser.add_argument('-o', dest='opt', type=str,
                        help='test option')
    return parser.parse_args(argv)


def test(parser, subparsers, module_name, argv, profile, log):
    args = test_args(add_parser(subparsers, module_name), subparsers, argv)
    if args.opt:
        greetings = args.opt
    else:
        greetings = 'Hello'
    print(" ".join(map(str, [greetings, args.test])))
```

An example of snippets using the Environment modules and the profile object:

### 5.1.2 Programmatic snippets access:

It is also possible to programmatically launch a snippet, within python console or within other python programs.

Although the interface it is not well documented yet.

```
import os
import argparse
os.environ['PYPE_SNIPPETS'] = 'test/data/snippets/'

from pype.modules import snippets

parser = argparse.ArgumentParser(prog='pype', description='Test')
subparsers = parser.add_subparsers(dest='modules')

snippets.snippets(None, subparsers, None, [
                  '--log', 'test/data/tmp', 'test',
                  '--test', 'World'], 'default')
snippets.snippets(None, subparsers, 'test', [
                  '--log', 'test/data/tmp', 'test',
                  '--test', 'World', '-o', 'Hello!'], 'default')
```

## 5.2 Pipelines

### 5.2.1 Writing new Pipelines:

The pipelines are YAML files located in a python module (mind the __init__.py in the folder containing the pipelines). The location of the module containing the pipelines can be controlled by the environment variable *PYPE_PIPELINES*.

The YAML file structure require a header field called *info* and the main field *items* with the execution information.

The *info* field expect the following keys:

1. description: with a brief description of the pipelines
2. date: with a string for tracking recent modification of the pipeline

3. arguments: which is not a required key, but it can be used to add information and description of the argument required in the pipeline

The *items* keys is constructed combining multiple blocks in a hierarchical manner. The items are called *PipelineItem*.

- The *PipelineItem* are block of information linked to execution of one or more tasks.
- A *PipelineItem* can wrap a snippet, another pipeline or the batch execution of a snippets or pipeline

The input of a batch snippet/pipeline is defined by the argument '–input_batch'. And consists in in a list of predefined arguments or tab separated file in which each line defines a run of the snippet/pipeline and each column refers to an argument of the snippet/pipeline (column names defines the specific argument).

Possible attributes in a *PipelineItem*:

- **Required:**
  - *type*: possible values: *snippets*, *pipeline*, *batch_snippets* or *batch_pipeline*
  - *name*: name of the item to look for in the specified *type* category.
  - *arguments*: specific arguments for the selected *type/name* function
- **Optional:**
  - *mute*: if *True* the PipelineItem will not return any results, hence it will not be listed as a dependencies of parents processes
  - *dependencies*: a list of other *PipelineItem*
  - *requirements*: dictionary to alter the default in-snippets requirements (apply only for *snippets* and *batch_snippets*)

An example of pipelines using the Environment modules and the profile object:

This will results in the command line interface:

```
pype pipelines bwa_mem

error: argument --qc_out is required
usage: pype pipelines bwa_mem --qc_out QC_OUT --tmp_dir TMP_DIR --bam_out
                              BAM_OUT --fq1 FQ1 --fq2 FQ2 --header HEADER

optional arguments:
  --qc_out QC_OUT    Path to store the QC output, type: str
  --tmp_dir TMP_DIR  Temporary directory, type: str
  --bam_out BAM_OUT  Resulting bam file, type: str
  --fq1 FQ1          First mate of fastq pairs, type: str
  --fq2 FQ2          Second mate of fastq pairs, type: str
  --header HEADER    @RG group header, comma separated, type: str
```

### 5.2.2 Programmatic pipelines access:

It is also possible to programmatically launch a pipeline, within python console or within other python programs.

As for the snippets, the interface it is not well documented yet.

```
import os
import argparse
os.environ['PYPE_SNIPPETS'] = 'test/data/snippets'
os.environ['PYPE_PIPELINES'] = 'test/data/pipelines'
```

(continues on next page)

(continued from previous page)

```
from pype.modules import pipelines

parser = argparse.ArgumentParser(prog='pype', description='Test')
subparsers = parser.add_subparsers(dest='modules')

input_fa = 'test/data/files/input.fa'
rev_fa = 'test/data/tmp/rev.fa'
compl_fa = 'test/data/tmp/rev_comp.fa'
out_fa = 'test/data/tmp/rev_comp_low.fa'
pipelines.pipelines(None, subparsers, None, [
          '--queue', 'none', '--log', 'test/data/tmp',
          'rev_compl_low_fa', '--input_fa', input_fa,
          '--reverse_fa', rev_fa, '--complement_fa',
          compl_fa, '--output', out_fa], 'default')
```

## 5.3 Profiles

### 5.3.1 Extending profiles:

Similarly to the pipelines, the profiles are YAML files in a python module (it is requires a __init__.py file in the folder containing the files). In order to be parsed correctly, the profile YAML file needs to have a specific structure:

```
info:
   description: Test Profile
   date:        14/03/2016

files:
   genome_fa:    /abs/path/to/fasta.fa
   genome_fa_gz: /abs/path/to/fasta.fa.gz

programs:
   samtools_0:
      path: samtools
      version: 0.1.19
   samtools_1:
      path: samtools
      version: 1.2
   bwa:
      path: bwa
      version: 0.7.10
   start:
      path: star
      version: 2.5.1b
```

CHAPTER 6

# Indexes and tables

- genindex
- modindex
- search

# Index