project_manager Documentation

Release 0.0.1

kpj

Nov 28, 2019

Contents

A utility which makes running the same projects with various configurations as easy as pie.

CHAPTER 1

Installation

```
$ pip install project_manager
```

CHAPTER 2

Usage

2.1 Configuration file

Create a configuration:

```
project_source: <url or path>  # project you want to run
working_dir: <path>  # where everything will run

exec_command:  # list of commands that will be executed in each project setup
    - <python ..>
result_files:  # list of files/folders that will be extracted after successful␣
↪execution
    - <result file>
    - <result dir>

base_config: <path>  # path to the raw configuration file (typically part of your␣
↪project)
symlinks:  # list of symlinks to include in each project setup
    - <path 1>
    - <path 2>
config_parameters:  # how to modify the configuration
    - key: param1
      values: [0, 1, 2]
      paired:
        - key: param2
          values: [a, b, c]
    - key: [nested, param3]
      values: ['a', 'b', 'c']
extra_parameters:  # special extra parameters
    git_branch: ['master']
    repetitions: 1
```

2.2 Commands

After setting up the configuration file, you can run all commands.

```
$ project_manager build
$ project_manager run
$ project_manager gather
```

In order, these commands do the following:

1. Create individual folders for each run and adapt the configuration accordingly
2. Run the specified commands per previously created setup
3. Retrieve all specified results into a single directory. Each individual files is annotated with its origin.

2.3 Example

This document provides a brief overview of `project_manager`'s basic functionality.

2.3.1 Environment setup

[2]:
```
tree
```

```
.
|____dummy_project
| |____my_conf.yaml
| |____run.py
|____config.yaml
```

[3]:
```
cat config.yaml
```

```
project_source: dummy_project
working_dir: tmp

exec_command:
    - python3 run.py
result_files:
    - results

base_config: dummy_project/my_conf.yaml
config_parameters:
    - key: message
      values: [A, B, C]
```

[4]:
```
cat dummy_project/my_conf.yaml
```

```
message: 'this is important'
```

[5]:
```
cat dummy_project/run.py
```

```
import os
import yaml
```

(continues on next page)

(continued from previous page)

```
def main():
    with open('my_conf.yaml') as fd:
        config = yaml.full_load(fd)

    os.makedirs('results')
    with open('results/data.txt', 'w') as fd:
        fd.write(config['message'])


if __name__ == '__main__':
    main()
```

2.3.2 Pipeline execution

Setup directory for each configuration

[6]:
```
project_manager build -c config.yaml
```

```
Setting up environments: 100%|| 3/3 [00:00<00:00, 880.60it/s]
```

Execute scripts for each configuration

[7]:
```
project_manager run -c config.yaml
```

```
  0%|                                                    | 0/3 [00:00<?, ?it/s]run.
↪message=A
 > python3 run.py
 33%|                           | 1/3 [00:00<00:00,  4.18it/s]run.message=C
 > python3 run.py
 67%|               | 2/3 [00:00<00:00,  4.61it/s]run.message=B
 > python3 run.py
100%|| 3/3 [00:00<00:00,  5.90it/s]
```

Gather results from each run

[8]:
```
project_manager gather -c config.yaml
```

```
run.message=A
 > results/data.txt
run.message=C
 > results/data.txt
run.message=B
 > results/data.txt
```

2.3.3 Investigate results

[9]:
```
tree tmp/
```

```
|____
|____aggregated_results
| |____results
| | |____data.message=B.txt
| | |____data.message=A.txt
| | |____data.message=C.txt
|____run.message=A
| |____my_conf.yaml
| |____results
| | |____data.txt
| |____run.py
|____run.message=C
| |____my_conf.yaml
| |____results
| | |____data.txt
| |____run.py
|____run.message=B
| |____my_conf.yaml
| |____results
| | |____data.txt
| |____run.py
```

[10]:
```
cat tmp/aggregated_results/results/*
```

```
ABC
```