digger-container-cli Documentation

Release 0.1

Red Hat

Sep 08, 2017

Contents

CHAPTER 1

Installation

Source code located at: https://github.com/aerogear/digger-build-cli

Requirements

Conda is recommended for environment setup: https://www.continuum.io/downloads

If running tests outside a container, *ANDROID_HOME* needs to be set as well.

Installing with pip

```
pip install git+ssh://git@github.com/aerogear/digger-build-cli.git
```

This will install a CLI script "abcd" together with the module itself.

Build from source

Clone the repository then run:

```
python setup.py install
```

Development

You can clone it and then run (considering conda is already installed):

```
conda env create -f env.yaml
```

To activate the dev environment run:

```
source activate digger
```

To run tests:

```
py.test -s
```

CHAPTER 2

API Reference

Actions

class digger.base.action.**Argument**(*name*, *flag*, ***kwargs*)
Class that stores arguments (argparse based) in BaseAction class

__init__(*name*, *flag*, ***kwargs*)
Argument class constructor, should be used inside a class that inherits the BaseAction class.

Parameters

- **name(str)** – the optional argument name to be used with two slahes (–cmd)
- **flag(str)** – a short flag for the argument (-c)
- ****kwargs** – all keywords arguments supported for argparse actions.

class digger.base.action.**BaseAction**
A class that should be subclassed to create a new action in the cli parser.

This action will become a subparser using its properties that subclass the class Argument as optional arguments for the subparser in context.

The subclass should define a _cmd_ and _help_ properties together with a instance handler method to receive the cli parameters.

cmd becomes the subparser command and _help_ is the help/info text about the command itself.

It also implements the __call__ method to be used as a function to parse the command args.

Example:

```
class FooAction(BaseAction):
  _cmd_ = 'foo'
  _help_ = 'some foo random action'

  Argument('--say', '-s', default='hello', help='say something, default value is␣
↪hello')

  def handler(self, say=None):
    print('Foo is saying: %s.' % say)
```

handler(***kwargs*)
Method to be overwrtten by the subclass to execute the actual command.

classmethod meta(*name*)
Used to get the _cmd_ and _help_ class properties.

Parameters **name(str)** – the property name to be retrieved.

Returns: The class properties that starts and ends with one underscore char based on the provided name.

classmethod props()
Class method that returns all defined arguments within the class.

Returns: A dictionary containing all action defined arguments (if any).

Commandline tool for container builds

CLI tool to build mobile apps inside a container

Errors

class digger.errors.**BaseError**(*message=None*)
Base exception class to be used whitin the module.

Every subclass error class should define an instance message property.

print_error()
Prints the error into teh STDOUT

class digger.errors.**InvalidPathError**(*path*, ***kwargs*)
Raised when an app folder path cannot be found.

class digger.errors.**InvalidZipFileError**(*path*, ***kwargs*)
Raised when the module can't unzip the app file (usually when the zipfile is corrupted).

class digger.errors.**BaseError**(*message=None*)
Base exception class to be used whitin the module.

Every subclass error class should define an instance message property.

print_error()
Prints the error into teh STDOUT

class digger.errors.**InvalidCMDError**(*cmd*, ***kwargs*)
Raised when and invalid command is used in the CLI.

class digger.errors.**MethodNotImplementedError**(***kwargs*)
Raised when a handler from a BaseAction subclass was not implemented.

class digger.errors.**DownloadError**(***kwargs*)
Raised when the module can't download the app source code from a given url.

class digger.errors.**InvalidProjectStructure**(***kwargs*)
Raised when the module can't build the app due to an invalid project structure. Example: gradlew file is missing from the gradle project.

Parser

digger.parser.**register_action**(*action*)
Adds an action to the parser cli.

Parameters **action(BaseAction)** – a subclass of the BaseAction class

digger.parser.**register_actions**(**args*)

Accepts N arguments to be added as parser actions.

Parameters ***args** – N number of actions that inherits from BaseAction.

digger.parser.**run**(**args*, ***kwargs*)

Runs the parser and it executes the action handler with the provided arguments from the CLI.

Also catches the BaseError interrupting the execution and showing the error message to the user.

Default arguments comes from the cli args (sys.argv array) but we can force those arguments when writing tests:

```
parser.run(['build', '--path', '/custom-app-path'].split())
```

```
parser.run('build --path /custom-app-path')
```

Builds

CHAPTER 3

Usage

Build

Executes the project build in a given path.

Args:

param –path the project root folder

Example:

```
abcd build --path /app
```

Inspect

Inspect the project file structure.

Args:

param –path the project root folder

Example:

```
abcd inspect --path /app
```

Export

Gets the build output file path (APK files for anroid).

Args:

param –path the project root folder

Example:

```
abcd export --path /app
```

Log

Reads the content of the build log files.

Args:

param –path the project root folder

param –ctx the log context (build, validate or all)

Example:

```
abcd log --path /app
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

d

Symbols

A

B

D

H

I

M

P

R