pmakeup

Release 1.0.0

Massimo Bono

Apr 30, 2021

CONTENTS:

CHAPTER

ONE

GETTING STARTED

1.1 Installation

To install pmakeup, just do

```
pip install pmakeup
```

To check if pmakeup has been successfully installed on your system, perform:

```
pmakeup --version
```

Calling *help* flag will show pmakeup help:

```
pmakeup --help
```

To install some other cool plugins, look for any package with *pmakeup-plugin*. For instance,

```
pip install archive-pmakeup-plugin
```

1.2 Uninstall

To uninstall the software, just call:

```
pip uninstall pmakeup
```

CHAPTER

TWO

CREATE PMAKEUP SCRIPTS

One of the main uses of `pmakeup` is to basically execute a python script, usually called `PMakeupfile.py` using an automatically generated *context*: this context, called **pmakeup registry** allows you to use some useful python functions and variables in your python script, without having to write anything. You don't need to import anything in your script.

If you need a function that is not available in the `pmakeup` project, you can either use or write a `pmakeup` plugin. If another developer has written a `pmakeup` plugin, you can just use the plugin by simply installing the package in your `pip` environment (`venv` are supported as well).

You can find some example of usage of `pmakeup` plugin in the `example/` folder (see *here https://github.com/Koldar/pmakeup/tree/main/examples*>)

2.1 Integrate pycharm in PMakeupfile

This step is completely optional. If you use an IDE, like Pycharm, you might want to use a content assist to help you writing the script.

To do so, import pmakeup file and then get the plugin instances you want to use. For instance:

```
import pmakeup as pm

core: pm.CorePMakeupPlugin = pmakeup_info.pmakeup_plugins[
↪"CorePMakeupPlugin"]
files: pm.FilesPMakeupPlugin = pmakeup_info.pmakeup_plugins[
↪"FilesPMakeupPlugin"]
log: pm.LoggingPMakeupPlugin = pmakeup_info.pmakeup_plugins[
↪"LoggingPMakeupPlugin"]
operating_system: pm.OperatingSystemPMakeupPlugin = pmakeup_info.
↪pmakeup_plugins["OperatingSystemPMakeupPlugin"]
paths: pm.PathsPMakeupPlugin = pmakeup_info.pmakeup_plugins[
↪"PathsPMakeupPlugin"]
targets: pm.TargetsPMakeupPlugin = pmakeup_info.pmakeup_plugins[
↪"TargetsPMakeupPlugin"]
```

The strings in the dictionary `pmakeup_plugins` is called **plugin name** and is customizable in the plugin class definition by overriding the method `get_plugin_name(self)` After this, you can write, for instance, `paths.` and the content assist should show all the available commands.

2.2 Simple PMakeup script

Usually the first thing you should write in your `PMakeupfile.py` is the *require_pmakeup_version*: in this way pmakeup can check if the currently installed version of pmakeup supports your script.

```
require_pmakeup_version("2.5.24")
```

After that, you can write basically anything you want by exploiting python.

2.3 PMakeup makefile-like

Albeit `pmakeup` is heavily inspired by makefile, its syntax is not very similar to it. At its base the section of `pmakeup` that implements a makefile style is the plugin `TargetsPMakeupPlugin`. At high level, we have a directed acyclic graph where each vertex presents a pmakeup target (i.e., a phony makefile target). Each directed edge represents the fact that in order to execute a target, you first need to execute its successors. Each vertex has a name, which is the target name, and a python function which takes no input and has no outputs, which is what pmakeup will execute whenever it is detected that such a function needs to be called.

In order to build the graph, you can code a `PMakeupfile` like this:

```
core: pm.CorePMakeupPlugin = pmakeup_info.pmakeup_plugins[
↪"CorePMakeupPlugin"]
log: pm.LoggingPMakeupPlugin = pmakeup_info.pmakeup_plugins[
↪"LoggingPMakeupPlugin"]
targets: pm.TargetsPMakeupPlugin = pmakeup_info.pmakeup_plugins[
↪"TargetsPMakeupPlugin"]

core.require_pmakeup_version("2.8.0")

def sayHello():
    log.print_blue("Hello")

def sayGoodbye():
    log.print_blue("And goodbye!")

targets.declare_file_descriptor(f"""
    A string that is used to describe what this script does
```

(continues on next page)

(continued from previous page)

```
""")
targets.declare_target(
    target_name="hello",
    description="pmakeup will say hello",
    f=sayHello,
    requires=[],
)
targets.declare_target(
    target_name="goodbye",
    description="Say goodbye after saying hello",
    f=sayGoodbye,
    requires=["hello"],
)

targets.process_targets()
```

This `PMakeupfile` does nothing and is pretty easy, but basically tells you the fundamentals of `PMakeupfile` targets.

You first need to define the functions corresponding to the targets (i.e., `sayHello` and `sayGoodbye`). Then you can possibly call `declare_file_descriptor` to improve the help information of the `pmakeup` script. After that, you need to write several `declare_target` function calls, one per graph vertex. The order is not important. you need to define the string that you need to input in order to call the corresponding function (*target_name*), the function that `pmakeup` needs to call whenever the target is requested (*sayHello*), a descrption to automatically build the help information (*description*) and finally the target dependencies (*requires*). The dependencies are an **ordered list** and there the order matters: putting a dependency near the head of the list means that the dependency is executeed before the others.

To invoke the help script, you can use `info`:

```
pmakeup --info
```

`pmakeup` will automatically show all the information you need to interact with the script. TO invoke the script, do the following:

```
pmakeup goodbye
```

Notice that in this case we will first invoke `hello` and only then we execute `goodbye`: this is due to the fact that `hello` is actually a requirements to `goodbye`.

CHAPTER

THREE

PMAKEUP AUTOLOADED PLUGINS

By default pmakeup has automatically loaded some plugin that allows you to make basic stuff (read from file, set variables). Such plugins are described here:

3.1 Core

class pmakeup.plugins.core.CorePMakeupPlugin.**CorePMakeupPlugin**(*model: pmakeup.models.P*

Bases: pmakeup.plugins.AbstractPmakeupPlugin.AbstractPmakeupPlugin

Contains all the commands available for the user in a PMakeupfile.py file

add_or_update_variable_in_cache(*name: str*, *supplier: Callable[], Any]*, *mapper: Callable[[Any], Any]*)

Add a new variable in the cache

Parameters

- **name** – the variable to set
- **supplier** – function used to generate the value fo the variable if the variable does not exist in the cache
- **mapper** – function used to generate the value fo the variable if the variable does exist in the cache. The input is the variable old value

clear_cache()

Clear the cache of pmakeup

ensure_condition(*condition: Callable[], bool]*, *message: str = ''*) → None

Perform a check. If the condition is **not** satisfied, we raise exception

Parameters

- **condition** – the condition to check. generate exception if the result is False

- **message** – the message to show if the exception needs to be generated

ensure_has_cli_variable(*name: str*) → None
Ensure the user has passed a variable via "–variable" CLI utils. If not, an exception is generated

Parameters **name** – the variable name to check

ensure_has_cli_variable_is_one_of(*name: str*, **allowed_values*) → None
Ensure that a variable has been passed from the command line and has a value among the one passed

Parameters

- **name** – variable name
- **allowed_values** – set of values we check against the variable calue

ensure_has_variable(*name: str*) → None
Ensure the user has passed a variable in the registry. If not, an exception is generated

Parameters **name** – the variable name to check

get_all_available_command_names() → Iterable[str]
Get all the commands you can execute right now

get_all_registered_plugins() → Iterable[str]
get all the registered pmakeup plugins at this moment

get_architecture() → int
check if the system is designed on a 32 or 64 bits

Returns either 32 or 64 bit

get_command_line_string() → str
Get the command line string from the user

Returns argv

get_home_folder() → str
Get the home fodler of the currently logged used

get_latest_path_with_architecture(*current_path: str*, *architecture: int*) → str
get the latest path on the system with the specified archietcture

Parameters

- **current_path** – nominal path name
- **architecture** – either 32 or 64

Returns the first path compliant with this path name

get_latest_version_in_folder(*folder: str = None*, *should_consider: Callable[[str], bool] = None*, *version_fetcher: Callable[[str], semantic_version.base.Version] = None*) → Tuple[semantic_version.base.Version, List[str]]

Scan the subfiles and subfolder of a given directory. We assume each file or folder has a version withint it. Then fetches the latest version. This command is useful in dierctories where all releases of a given software are placed. if we need to fetch the latest one, this function is perfect for the task.

Parameters

- **folder** – the folder to consider. If unspecified, it is the current working directory
- **should_consider** – a function that allows you to determine if we need to consider or not a subfile/subfolder. The input isan absolute path. If no function is given, we accept all the sub files
- **version_fetcher** – a function that extract a version from the filename. If left unspecified, we will use ::semantic_version_2_only_core

Returns the latest version in the folder. The second element of the tuple is a collection of all the filenames that specify the latest version

get_pmakeupfile_dir() → str

The directory where the analyzed pmakeupfile is located

Returns absolute ptha of the directory of the path under analysis

get_pmakeupfile_dirpath() → str

Returns absolute path of the folder containing the main PMakeupfile path

get_pmakeupfile_path() → str

Returns absolute path of the main PMakeupfile path

get_starting_cwd() → str

Returns absolute path of where you have called pmakeup

get_variable_in_cache(*name: str*) → Any

Get the variable from the cache. if the variable does not exist, an error is generated

Parameters **name** – name of the variable to check

Returns the value associated to such a variable

get_variable_in_cache_or(*name: str*, *default: Any*) → Any

Get the variable value from the cache or get a default value if it does not exist

Parameters

- **name** – name of the variable to fetch
- **default** – if the variable does not exist in the cache, the value to retturn from this function

Returns the variable value

get_variable_in_cache_or_fail(*name: str*) → Any
Get the variable value from the cache or raise an error if it does not exist

Parameters **name** – name of the variable to fetch

Returns the variable value

has_variable_in_cache(*name: str*) → bool
Check if a variable is in the pmakeup cache

Parameters **name** – name of the variable to check

Returns true if a varaible with such a name is present in the cache, false otherwise

include_file(**file: str*) → None
Replace the include directive with the content fo the included file. Fails if there is no such path

Parameters **file** – the external file to include in the script

include_string(*string: str*) → None
Include and execute the code within the given string

Parameters **string** – the commands to execute

is_process_running(*program_name: str*) → bool
Check if a program with the given name is currently running

Parameters **program_name** – the program we need to check

Returns true if we are running such a program, false otheriwse

kill_process_by_name(*program_name: str*, *ignore_if_process_does_not_exists: bool = True*)
Kill a program

Parameters

- **program_name** – name fo the program that is running on the system
- **ignore_if_process_does_not_exists** – if the proces does not exist and thsi parameter is true, this function will **not** throw exception

kill_process_by_pid(*pid: int*, *ignore_if_process_does_not_exists: bool = True*)
Kill a program

Parameters

- **pid** – pid of the program that is running on the system
- **ignore_if_process_does_not_exists** – if the proces does not exist and thsi parameter is true, this function will **not** throw exception

load_cache()
Load all the variables present in cache into the available variables

log_command(*message: str*)
reserved. Useful to log the action performed by the user

Parameters **message** – message to log

on_linux() → bool
Check if we are running on linux

Returns true if we are running on linux

on_windows() → bool
Check if we are running on windows

Returns true if we are running on windows

path_wrt_pmakeupfile(**folder: str*) → str
Compute path relative to the file where PMakeupfile is located

Parameters **folder** – other sections of the path

Returns path relative to the absolute path of where PMakeupfile is located

path_wrt_starting_cwd(**folder: str*) → str
Compute path relative to the starting cwd

Parameters **folder** – other sections of the path

Returns path relative to the absolute path of where you have called pmakeup

quasi_semantic_version_2_only_core(*filename: str*) → semantic_version.base.Version
A function that can be used within ::get_latest_version_in_folder. It accepts values like "1.0.0", but also "1.0" and "1"

Parameters **filename** – the absolute path of a file that contains a version

Returns the version

read_variables_from_properties(*file: str*, *encoding: str = 'utf-8'*) → None
Read a set of easy variables from a property file. All the read variables will be available in the "variables" value. If some variable name preexists, it will not be overriden :see: https://docs.oracle.com/cd/E23095_01/Platform.93/ATGProgGuide/html/s0204propertiesfileformat01.html

Parameters

- **file** – the file to read
- **encoding** – encoding of the file. If left missing, we will use utf-8

require_pmakeup_plugins(**pmakeup_plugin_names: str*)

Tells pmakeup that, in order to run the script, you required a sequence of pmakeup plugins correctly installed (the version does not matter)

Pmakeup will then arrange itself in installing dependencies and the correct order of the plugins

Parameters **pmakeup_plugin_names** – the plugins that are requierd to be present in order for the script to work. Dependencies are automatically added

require_pmakeup_version(*lowerbound: str*) → None

Check if the current version of pmakeup is greater or equal than the given one. If the current version of pmakeup is not compliant with this constraint, an error is generated

Parameters **lowerbound** – the minimum version this script is compliant with

semantic_version_2_only_core(*filename: str*) → semantic_version.base.Version

A function that can be used within ::get_latest_version_in_folder

Parameters **filename** – the absolute path of a file that contains a version

Returns the version

set_variable_in_cache(*name: str*, *value: Any*, *overwrite_if_exists: bool = True*)

Set a variable inside the program cache. Setting variable in cache allows pmakeup to store information between several runs of pmakeup.

How pmakeup stores the information is implementation dependent and it should not be relied upon

Parameters

- **name** – name of the variable to store
- **value** – object to store
- **overwrite_if_exists** – if true, if the cache already contain a variable with the same name, such a varaible will be replaced with the new one

vars() → pmakeup.models.AttrDict.AttrDict

Get a dictioanry containing all the variables setup up to this point. You can use thi dictionary to gain access to a variable in a more pythonic way (e.g., vars.foo rather than get_variable("foo")

Raises ***PMakeupException*** – if the variable is not found

3.2 Files

class pmakeup.plugins.files.FilesPMakeupPlugin.**FilesPMakeupPlugin**(*model: pmakeup.mod*

Bases: pmakeup.plugins.AbstractPmakeupPlugin.AbstractPmakeupPlugin

allow_file_to_be_executed_by_anyone(*file: str*)

Allow the file to be executed by anyone. On a linux system it should be equal to "chmod o+x"

Parameters **file** – the file whose permission needs to be changed

append_string_at_end_of_file(*name: str*, *content: Any*, *encoding: str = 'utf-8'*) → None

Append a string at the end of the file. carriage return is automatically added

Parameters

- **name** – filename
- **content** – string to append
- **encoding** – encoding of the file. If missing, "utf-8" is used

append_strings_at_end_of_file(*name: str*, *content: Iterable[Any]*, *encoding: str = 'utf-8'*) → None

Append a string at the end of the file. carriage return is automatically added

Parameters

- **name** – filename
- **content** – string to append
- **encoding** – encoding of the file. If missing, "utf-8" is used

copy_file(*src: str*, *dst: str*, *create_dirs: bool = True*)

Copy a single file from a position to another one. If the destination folder hierarchy does not exist, we will create it

Parameters

- **src** – file to copy
- **dst** – destination where the file will be copied to. If a file, we will copy the src file into another file with different name. If a directory, we will copy the specified file into the dirctory dst (without altering the filename)

- **create_dirs** – if true, we will create the directories of dst if non existent

copy_files_that_basename(*src: str*, *dst: str*, *regex: str*)

Copy the files located (directly or indirctly) in src into dst. We will copy only the files whose basename (e.g. foo.txt is the basename of /opt/foo/bar/foo.txt). We will copy the directories where a file is located as well matches the given regex

Parameters

- **src** – folder where we will find files to copy
- **dst** – destination of the files
- **regex** – regex that determines wether or not a file is copies

Returns

copy_folder_content(*folder: str*, *destination: str*)

Copy all the content of "folder" into the folder "destination"

Parameters

- **folder** – folder to copy files from
- **destination** – folder where the contents will be copied into

copy_tree(*src: str*, *dst: str*)

Copy a whole directory tree or a single file. If you specifies a file rather than a directory, the function behaves like :see copy_file

Parameters

- **src** – the folder or the file to copy.
- **dst** – the destination where the copied folder will be positioned

create_empty_directory(*name: str*) → str

Create an empty directory in the CWD (if the path is relative)

:param name:the name of the driectory to create :return: the full path of the directory just created

create_empty_file(*name: str*, *encoding: str = 'utf-8'*)

Create an empty file. if the file is relative, it is relative to the CWD

Parameters

- **name** – file name to create
- **encoding** – encoding of the file. If unspecified, it is utf-8

find_directory(*root_folder: str*, *folder: str*) → Iterable[str]

Find all the directories with the given name

Parameters

- **root_folder** – fodler where we need to look int
- **folder** – name of the folder we need to fetch

Returns list of files with thwe given filename

find_directory_with_filename_compliant_with_regex(*root_folder: str*, *folder_regex: str*) → Iterable[str]

Find all the directories with the given name

Parameters

- **root_folder** – fodler where we need to look int
- **folder_regex** – regex the folder name should be compliant with

Returns list of files with thwe given filename

find_directory_with_fullpath_compliant_with_regex(*root_folder: str*, *folder_regex: str*) → Iterable[str]

Find all the directories with the given name

Parameters

- **root_folder** – folder where we need to look int
- **folder_regex** – regex the folder name should be compliant with

Returns list of files with thwe given filename

find_executable_in_program_directories(*program_name: str*, *fail_if_program_is_not_found: bool = False*) → Optional[str]

Find a program ouside the path as well. Paths is still considered

Parameters

- **program_name** – name of the program to look for
- **fail_if_program_is_not_found** – if true, we will raise an exception if the program is not found

Returns first absolute path of the program found. None if we did not find the program

find_file(*root_folder: str*, *filename: str*) → Iterable[str]

Find all the files with the given filename (extension included)

Parameters

- **root_folder** – fodler where we need to look int
- **filename** – filename we need to fetch

Returns list of files with thwe given filename

find_file_in_roots_st(*root_folders: str*, *match: Callable[[str, str, str], bool]*) → Iterable[str]

Find all the files matchign the given function

Parameters

- **root_folders** – folders where we need to look int
- **match** – a function that defines if you want to include the file into the output. The first parameter is the folder containing the given file. The second parameter is the involved file. The third is the absolute path of the involved path

Returns list of files compliant with the given function

find_file_st(*root_folder: str*, *match: Callable[[str, str, str], bool]*) → Iterable[str]

Find all the files matchign the given function

Parameters

- **root_folder** – folder where we need to look int
- **match** – a function that defines if you want to include the file into the output. The first parameter is the folder containing the given file. The second parameter is the involved file. The third is the absolute path of the involved path

Returns list of files compliant with the given function

find_file_with_filename_compliant_with_regex(*root_folder: str*, *filename_regex: str*) → Iterable[str]

Find all the files containign (search) the given regex

Parameters

- **root_folder** – folder where we need to look int
- **filename_regex** – the regex any filename should be compliant

Returns list of files with thwe given filename

find_file_with_fullpath_compliant_with_regex(*root_folder: str*, *filename_regex: str*) → Iterable[str]

Find all the files containing (search) the given regex

Parameters

- **root_folder** – folder where we need to look int
- **filename_regex** – the regex any filename should be compliant

Returns list of files with the given filename

find_first_file_in_roots_st(*root_folders: str*, *match: Callable[[str, str, str], bool]*) → Optional[str]

Find the first file matching the given function

Parameters

- **root_folders** – folders where we need to look int
- **match** – a function that defines if you want to include the file into the output. The first parameter is the folder containing the given file. The second parameter is the involved file. The third is the absolute path of the involved path

Returns file compliant with the given function or None

find_first_file_in_roots_st_or_fail(*root_folders: str*, *match: Callable[[str, str, str], bool]*) → str

Find the first file matching the given function. If no such file exists, generates an exception

Parameters

- **root_folders** – folders where we need to look int
- **match** – a function that defines if you want to include the file into the output. The first parameter is the folder containing the given file. The second parameter is the involved file. The third is the absolute path of the involved path

Returns file compliant with the given function or None

find_first_file_st(*root_folder: str*, *match: Callable[[str, str, str], bool]*) → Optional[str]

Find the first file matching the given function

Parameters

- **root_folder** – folder where we need to look int

- **match** – a function that defines if you want to include the file into the output. The first parameter is the folder containing the given file. The second parameter is the involved file. The third is the absolute path of the involved path

Returns file compliant with the given function or None

find_first_file_st_or_fail(*root_folder: str*, *match: Callable[[str, str, str], bool]*) → str

Find the first file matching the given function. If no such file exists, generates an exception

Parameters

- **root_folder** – folder where we need to look int
- **match** – a function that defines if you want to include the file into the output. The first parameter is the folder containing the given file. The second parameter is the involved file. The third is the absolute path of the involved path

Returns file compliant with the given function or None

find_folder_st(*root_folder: str*, *match: Callable[[str, str, str], bool]*) → Iterable[str]

Find all the folder matching a given function

Parameters

- **root_folder** – folder where we need to look int
- **match** – a function that defines if you want to include the folder into the output. The first parameter is the folder containing the given folder. The second parameter is the involved folder. The third is the absolute path of the involved path

Returns list of folders compliant with the given function

find_regex_match_in_file(*pattern: str*, **p: str*, *encoding: str = 'utf8'*, *flags: Union[int, re.RegexFlag] = 0*) → Optional[re.Match]

FInd the first regex pattern in the file

If you used named capturing in the pattern, you can gain access via result.group("name")

Parameters

- **pattern** – regex pattern to consider
- **p** – file to consider
- **encoding** – encoding of the file to search. Defaults to utf8

- **flags** – flags of the regex to build. Passed as-is

Returns a regex match representing the first occurence. If None we could not find anything

get_file_size(**f: str*) → int
Get the filesize of a given file. If the file is a directory, return the cumulative size of all the files in it

Parameters **f** – the path of the file to consider

Returns number of bytes

is_directory(**p: str*) → bool
Check if the given path is a directory

Parameters **p** – paths to check

Returns true if the concatenated version of p is a directory. False otherwise

is_directory_empty(*name: str*) → bool
Check if a directory exists and is empty

Parameters **name** – folder to check

Returns true if the folder exists and is empty, false otherwise

is_directory_exists(*name: str*) → bool
Check if a directory exists.

Parameters **name** – folder to check

Returns true if the folder exists, false otherwise

is_file(**p: str*) → bool
Check if the given path represents a file or a directory

Parameters **p** – paths to check

Returns true if the concatenated version of p is a file. False otherwise

is_file_empty(*name: str*) → bool
Checks if a file exists. If exists, check if it empty as well.

Parameters **name** – file to check

Returns true if the file exists **and** has no bytes; false otherwise

is_file_exists(*name: str*) → bool
Check if a file exists

Parameters **name** – file whose existence we need to assert

Returns true if the file exists, false otherwise

is_file_non_empty (**name: str*) → bool
Checks if a file exists. If exists, check if it is not empty as well.

Parameters **name** – file to check

Returns true if the file exists **and** has at least one byte; false otherwise

ls (*folder: str = None*, *generate_absolute_path: bool = False*) → Iterable[str]
Show the list of all the files in the given directory

Parameters

- **folder** – folder to scan. default to CWD
- **generate_absolute_path** – if true, we will generate in the outptu the absolute path of the subfolders. Otherwise we will return only the

Returns iterable of all the files in the given directory

ls_directories_recursive (*folder: str*) → Iterable[str]
Show the list of all the directories in the given folder

Parameters **folder** – folder to scan (default to cwd)

Returns list of absolute filename representing the stored directories

ls_only_directories (*folder: str = None*, *generate_absolute_path: bool = False*) → Iterable[str]
Show the list of all the directories in the given directory

Parameters

- **folder** – folder to scan. If missing, default to CWD
- **generate_absolute_path** – if true, we will generate in the outptu the absolute path of the subfolders. Otherwise we will return only the names.

Returns a list of absolute paths representing the subdirectories inside `folder`

ls_only_files (*folder: str = None*, *generate_absolute_path: bool = False*) → Iterable[str]
Show the list of all the files (but not directories) in the given directory

Parameters

- **folder** – folder to scan. default to CWD
- **generate_absolute_path** – if true, we will generate in the outptu the absolute path of the subfolders. Otherwise we will return only the

Returns

ls_recursive(*folder: str = None*) → Iterable[str]
Show the list of all the files in the given folder

Parameters **folder** – folder to scan (default to cwd)

Returns list of absolute filename representing the stored files

make_directories(**folder: str*) → None
Create all the needed directories for the given path. Note that if you inject the path *temp/foo/hello.txt* (you can see hello.txt shoudl be a file) the function will generate hello.txt as a **directory**!

Parameters **folder** – folders to create

move_file(*src: str*, *dst: str*)
Move a single file from a location to another one

Parameters

- **src** – the file to move
- **dst** – the path where the file will be moved to

move_tree(*src: str*, *dst: str*)
Move an entire directory tree from one position to another one

Parameters

- **src** – path of the directory to move
- **dst** – path of the directory that we will create

read_file_content(*name: str*, *encoding: str = 'utf-8'*, *trim_newlines: bool = True*) → str
Read the whole content of the file in a single string

Parameters

- **name** – name of the file to load
- **encoding** – the encoding of the file. If unspecified, it is utf-8
- **trim_newlines** – if true, we will trim the newlines, spaces and tabs at the beginning and at the end of the file

Returns string repersenting the content of the file

read_lines(*name: str*, *encoding: str = 'utf-8'*) → Iterable[str]
Read the content of a file and yields as many item as there are lines in the file. Strip from the line ending new lines. Does not consider empty lines

Parameters

- **name** – name of the file
- **encoding** – encoding of the file. If unspecified, it is utf-8

Returns iterable containing the lines of the file

remove_file(*name: str*, *ignore_if_not_exists: bool = True*) → bool

Remove a file. If the cannot be removed (for some reason), ignore_if_not_exists determines if somethign goes wrong

Parameters

- **name** – file to delete
- **ignore_if_not_exists** – if true, we won't raise exception if the file does not exists or cannot be removed

Returns true if we have removed the file, false otherwise

remove_files_that_basename(*src: str*, *regex: str*)

Remove the files located (directly or indirectly) in src. We will copy only the files whose basename (e.g. foo.txt is the basename of /opt/foo/bar/foo.txt). We will copy the directories where a file is located as well matches the given regex

Parameters

- **src** – folder where we will find files to copy
- **regex** – regex that determines wether or not a file is copies

Returns

remove_last_n_line_from_file(*name: str*, *n: int = 1*, *consider_empty_line: bool = False*, *encoding: str = 'utf-8'*) → List[str]

Read the content of a file and remove the last n lines from the file involved. Then, rewrites the whole file

Parameters

- **name** – file involved. If relative, it is relative to ::cwd()
- **n** – the number of lines to remove at the end.
- **consider_empty_line** – if True, we consider empty lines as well.
- **encoding** – the encoding used to rewrite file

Returns the lines just removed

remove_string_in_file(*name: str*, *substring: str*, *count: int = - 1*, *encoding: str = 'utf-8'*)

Remove some (or all) the occurences of a given substring in a file

Parameters

- **name** – path of the file to handle
- **substring** – substring to replace

- **count** – the number of occurences to remove. -1 if you want to remove all occurences
- **encoding** – encoding used for reading the file

remove_tree(**folder: str*, *ignore_if_not_exists: bool = True*) → None

Remove a dirctory tree

Parameters

- **folder** – path to the directory to remove
- **ignore_if_not_exists** – if the directory does not exists, we do nothing if htis field is true

replace_regex_in_file(*name: str*, *regex: str*, *replacement: str*, *count: int = -1*, *encoding: str = 'utf-8'*)

Replace some (or all) the occurences of a given regex in a file.

If you want to use named capturing group, you can do so! For instance,

replace_regex_in_file(file_path, '(?P<word>w+)', '(?P=word)aa') 'spring' will be replaced with 'springaa'

It may not work, so you can use the following syntax to achieve the same: replace_regex_in_file(file_path, '(?P<word>w+)', r'g<word>aa') 'spring' will be replaced with 'springaa'

Parameters

- **name** – path of the file to handle
- **regex** – regex to replace
- **replacement** – string that will replace *substring*
- **count** – the number of occurences to replace. -1 if you want to replace all occurences
- **encoding** – encoding used for reading the file

See https://docs.python.org/3/howto/regex.html

replace_string_in_file(*name: str*, *substring: str*, *replacement: str*, *count: int = - 1*, *encoding: str = 'utf-8'*)

Replace some (or all) the occurences of a given substring in a file

Parameters

- **name** – path of the file to handle
- **substring** – substring to replace
- **replacement** – string that will replace *substring*

- **count** – the number of occurences to replace. -1 if you want to replace all occurences
- **encoding** – encoding used for reading the file

write_file(*name: str*, *content: Any*, *encoding: str = 'utf-8'*, *overwrite: bool = False*, *add_newline: bool = True*)

Write into a file with the specified content. if overwrite is unset, we will do nothing if the file already exists

Parameters

- **name** – name of the file to create
- **content** – content of the file to create.
- **encoding** – encoding fo the file to create. utf-8 by default
- **overwrite** – if true, we will overwrite the file
- **add_newline** – if true, we will add a new line at the end of the content

write_lines(*name: str*, *content: Iterable[Any]*, *encoding: str = 'utf-8'*, *overwrite: bool = False*)

Write severla lines into a file. if overwrite is unset, we will do nothing if the file already exists

Parameters

- **name** – name of the file to create
- **content** – lines of the file to create. We will append a new ine at the end of each line
- **encoding** – encoding fo the file to create. utf-8 by default
- **overwrite** – if true, we will overwrite the file

3.3 Paths

class pmakeup.plugins.paths.PathsPMakeupPlugin.**PathsPMakeupPlugin**(*model: pmakeup.mod*

Bases: pmakeup.plugins.AbstractPmakeupPlugin.AbstractPmakeupPlugin

abs_path(**p:* pmakeup.plugins.paths.PathsPMakeupPlugin.PathsPMakeupPlugin.path) → str

Generate a path compliant with the underlying operating system path scheme.

If the path is relative, it is relative to the cwd

Parameters **p** – the path to build

cd(**folder: str*, *create_if_not_exists: bool = True*) → str

Gain access to a directory. If the directory does nto exists, it is created If the path is relative, it is relative to the CWD

Parameters

- **folder** – folder where we need to go into
- **create_if_not_exists** – if true, we will create the directory if we try to cd into a non existent directory

Returns the directory where we have cd from

cd_into_directories(*folder: str*, *prefix: str*, *folder_format: str*, *error_if_mismatch: bool = True*)

Inside the given folder, there can be several folders, each of them with the same format. We cd into the "latest" one. How can we determine which is the "latest" one? Via folder_format. it is a string that is either: - "number": an integer number - "semver2": a semantic versionign string; We fetch the "latest" by looking at the one with the greater value. If the folder contains a folder which it is not compliant with folder_format, it is either ignored or rase error

Parameters

- **folder** – folder where several folders are located
- **prefix** – a string that prefix folder_format
- **folder_format** – either "number" or "semver2"
- **error_if_mismatch** – if a folder is not compliant with folder_format, if true we will generate an exception

Returns

change_filename_extension(*new_extension: str*, **p*) → str

Change the extension of the given path

new extensions: dat /path/to/file.txt.zp.asc -> /path/to/file.txt.zp.dat

Parameters

- **new_extension** – extension that will be set
- **p** – path to chan

Returns p, but with the updated extensions

cwd() → str

Returns the CWD the commands operates in

get_absolute_file_till_root(*filename: str*, *base: str = None*) → str

Starting from the directory base, check if a fiel called "filename" is present. If nt,

recursively chekc the parent directory Raise an exception if the file is not found whern considering the root

Parameters

- **filename** – the name of the file (extension included) we need to look for
- **base** – directory where we start looking. If left missing, we consider the CWD

Returns absolute path of the file found

get_basename(*p) → str
Compute the base name of the path

/path/to/file.txt.zp.asc -> file.txt.zp.asc

Parameters **p** – path to consider

Returns basename

get_basename_with_no_extension(*p) → str
Compute the basename of the path and remove its extension as well

/path/to/file.txt.zp.asc -> file.txt.zp

Parameters **p** – path to consider

Returns basename

get_extension(*p) → str
Compute the extension of a file

Parameters **p** – the file to consider

Returns the file extension

get_file_without_extension(*p: str*) → str
Compute the filename without its last extension

/path/to/some/file.txt.zip.asc –> /path/to/some/file.txt.zip

Parameters **p** – path to consider

Returns same absolute path, without extension

get_parent_directory(*p) → str
Retrieve the absolute path of the parent directory of the specified path.

/foo/tbar/tmp.txt -> /foo/tbar

Parameters **p** – path to consider

Returns parent directory of path

get_relative_path_wrt (*p: str*, *reference: str*) → str

If we were in folder reference, what actiosn should we perform in order to reach the file p?

Parameters

- **p** – the file to reach
- **reference** – the folder we are in right now

Returns relative path

path (**p: str*) → str

Generate a path compliant wit the underlying operating system path scheme.

If the path is relative, we will **not** join it with cwd

Parameters **p** – the path to build

3.4 Strings

class pmakeup.plugins.strings.StringsPMakeupPlugin.**StringsPMakeupPlugin**(*mode*
pmak

Bases: pmakeup.plugins.AbstractPmakeupPlugin.AbstractPmakeupPlugin

match (*string: str*, *regex: str*) → bool

Check if a given string matches perfectly the given regex

Parameters

- **string** – the sting to check
- **regex** – the regex to check. The syntax is available at https://docs.python.org/3/library/re.html

Returns true if such a substring can be found, false otherwise

replace_regex_in_string (*string: str*, *regex: str*, *replacement: str*, *count: int = - 1*, *encoding: str = 'utf-8'*) → str

Replace some (or all) the occurences of a given string

If you want to use named capturing group, you can do so! For instance,

replace_regex_in_string('3435spring9437', r'(?P<word>[a-z]+)', r'aa') 'spring' will be replaced with 'springaa'

It may not work, so you can use the following syntax to achieve the same: replace_regex_in_file(file_path, '(?P<word>w+)', r'g<word>aa') 'spring' will be replaced with 'springaa'

Parameters

- **string** – string that will be involved in the replacements
- **regex** – regex to replace
- **replacement** – string that will replace *substring*
- **count** – the number of occurences to replace. -1 if you want to replace all occurences
- **encoding** – encoding used for reading the file

See https://docs.python.org/3/howto/regex.html

replace_substring_in_string(*string: str*, *substring: str*, *replacement: str*, *count: int = - 1*) → str

Replace some (or all) the occurences of a given string

Parameters

- **string** – string that will be involved in the replacements
- **substring** – the string to repplace
- **replacement** – string that will replace *substring*
- **count** – the number of occurences to replace. -1 if you want to replace all occurences

search(*string: str*, *regex: str*)

Check if a given string has a substring that matches the given regex

Parameters

- **string** – the sting to check
- **regex** – the regex to check. The syntax is available at https://docs.python.org/3/library/re.html

Returns true if such a substring can be found, false otherwise

3.5 Targets

class pmakeup.plugins.targets.TargetsPMakeupPlugin.**TargetsPMakeupPlugin**(*mode*
pmak

Bases: pmakeup.plugins.AbstractPmakeupPlugin.AbstractPmakeupPlugin

declare_file_descriptor(*description: str*)

Defines what to write at the beginning of the info string that is displayed whenver the user wants to know what the given Pmakeupfile does

Parameters **description** – string to show

declare_target(*target_name: str*, *f: Callable[], None]*, *requires: Iterable[str] = None*, *description: str = ''*)
Declare that the user can declare a pseudo-makefile target.

Parameters

- **target_name** – name of the target to declare
- **description** – a description that is shown when listing all available targets
- **requires** – list fo target names this target requires in order to be executed. They must already exist in pmakeup environment
- **f** – the function to perform when the user requests this target

get_target_descriptor(*target_name: str*) → pmakeup.TargetDescriptor.TargetDescriptor
Get a descriptor for a given pmakeup target. Raises exception if target is not declared

Parameters **target_name** – name of the target

Returns descriptor for the target

is_target_requested(*target_name: str*) → bool
Check if the the user has specified the given target

Parameters **target_name** – the name of the target that we need to check

Returns true if the target has been declard by the user, false otherwise

process_targets()
Function used to process in the correct order. If the user requested to show the help for this file, the function will show it and return it

It will call the function declared in declare_target

3.6 TempFiles

class pmakeup.plugins.tempfiles.TempFilesPMakeupPlugin.**TempFilesPMakeupPlug**

Bases: pmakeup.plugins.AbstractPmakeupPlugin.AbstractPmakeupPlugin

create_temp_directory_with(*directory_prefix: str*) → Any
Create a temporary directory on the file system where to put temporary files

Parameters **directory_prefix** – a prefix to be put before the temporary folder

Returns the absolute path of the temporary folder created. The function can be used an input of a "with" statement. The folder will be automatically removed at the end of the with.

create_temp_file(*directory: str*, *file_prefix: str = None*, *file_suffix: str = None*, *mode: str = 'r'*, *encoding: str = 'utf-8'*, *readable_for_all: bool = False*, *executable_for_owner: bool = False*, *executable_for_all: bool = False*) → str

Creates the file. You need to manually dispose of the file by yourself

Parameters

- **directory** – the directory where to put the file
- **file_prefix** – a string that will be put at the beginning of the filename
- **file_suffix** – a string that will be put at the end of the filename
- **mode** – how we will open the file. E.g., "r", "w"
- **encoding** – the encodign of the file. Default to "utf-8"
- **readable_for_all** – if True, the file can be read by anyone
- **executable_for_owner** – if True, the file can be executed by the owner
- **executable_for_all** – if True, anyone can execute the file

Returns the absolute path of the temp file

get_temp_filepath(*prefix: str = None*, *suffix: str = None*) → str

Get the filename of a temp file. You need to manually create such a temp file

Parameters

- **prefix** – a prefix the temp file to generate has
- **suffix** – a suffix the temp file to generate has

Returns the absolute path of the temp path

3.7 Utils

class pmakeup.plugins.utils.UtilsPMakeupPlugin.**UtilsPMakeupPlugin**(*model: pmakeup.mod*

Bases: pmakeup.plugins.AbstractPmakeupPlugin.AbstractPmakeupPlugin

as_bool(*v: Any*) → bool

Convert a value into a boolean

Parameters **v** – value to convert as a boolean

Returns true of false

convert_table(*table_str: str*) → List[List[str]]
Convert a table printed as:

Port Type Board Name FQBN Core /dev/ttyACM1 Serial Port (USB) Arduino/Genuino MKR1000 arduino:samd:mkr1000 arduino:samd

Into a list of lists of strings

Parameters **table_str** – representation of a table

Returns list of lists of strings

get_column_of_table(*table: List[List[str]]*, *index: int*) → List[str]
Select a single column from the table, generated by ::convert_table

Parameters

- **table** – the table generated by ::convert_table
- **index** – index of the column to return. Starts from 0

Returns the column requested

get_column_of_table_by_name(*table: List[List[str]]*, *column_name: str*) → List[str]
Select a single column from the table, generated by ::convert_table We assumed the first row of the table is a header, contaiing the column names

Parameters

- **table** – the table generated by ::convert_table
- **column_name** – name of the column to return.

Returns the column requested

grep(*lines: Iterable[str]*, *regex: str*, *reverse_match: bool = False*) → Iterable[str]
Filter the lines fetched from terminal

Parameters

- **lines** – the lines to fetch
- **regex** – a python regex. If a line contains a substring which matches the given regex, the line is returned
- **reverse_match** – if True, we will return lines which do not match the pattern

Returns lines compliant with the regex

pairs (*it: Iterable[Any]*) → Iterable[Tuple[Any, Any]]
Convert the iterable into an iterable of pairs.

1,2,3,4,5,6 becomes (1,2), (2,3), (3,4), (4,5), (5,6)

Parameters **it** – iterable whose sequence we need to generate

Returns iterable of pairs

3.8 Logging

class pmakeup.plugins.log.LoggingPMakeupPlugin.**LoggingPMakeupPlugin**(*model: pmakeup.m*

Bases: pmakeup.plugins.AbstractPmakeupPlugin.AbstractPmakeupPlugin

critical (*message: str*)
Log a message using 'CRITICAL' level

Parameters **message** – the message to log

debug (*message: str*)
Log a message using 'DEBUG' level

Parameters **message** – the message to log

echo (*message: str*, *foreground: str = None*, *background: str = None*)
Print a message on the screen

Parameters

- **message** – the message to print out
- **foreground** – foreground color of the string. Accepted values: RED, GREEN, YELLOW, BLUE, MAGENT, CYAN, WHITE
- **background** – background color of the string. Accepted values: RED, GREEN, YELLOW, BLUE, MAGENT, CYAN, WHITE

echo_variables (*foreground: str = None*, *background: str = None*)
Echo all the variables defined in "variables"

Parameters

- **foreground** – the foregruodn color
- **background** – the background color

info (*message: str*)
Log a message using 'INFO' level

Parameters **message** – the message to log

print_blue(*message: str*)
Print a blue message

Parameters **message** – message to print

print_cyan(*message: str*)
Print a blue message

Parameters **message** – message to print

print_red(*message: str*)
Print a red message

Parameters **message** – message to print

print_yellow(*message: str*)
Print a blue message

Parameters **message** – message to print

3.9 Operating system

class pmakeup.plugins.operating_system.OperatingSystemPMakeupPlugin.**Operati**

Bases: pmakeup.plugins.AbstractPmakeupPlugin.AbstractPmakeupPlugin

current_user() → str
get the user currently logged

Returns the user currently logged

execute_admin_and_forget(*commands: Union[str, List[Union[str, List[str]]]], cwd: str = None, env: Dict[str, Any] = None, check_exit_code: bool = True, timeout: int = None*) → int
Execute a command as admin but ensure that no stdout will be printed on the console

Parameters

- **commands** – the command to execute. They will be exeucte in the same context
- **cwd** – current working directory where the command is executed
- **env** – a dictionary representing the key-values of the environment variables
- **check_exit_code** – if true, we will generate an exception if the exit code is different than 0

- **timeout** – if positive, we will give up waiting for the command after the amount of seconds

Returns triple. The first element is the error code, the second is the stdout (if captured), the third is stderr

execute_admin_and_run_in_background(*commands: Union[str, List[Union[str, List[str]]]]*, *cwd: str = None*, *env: Dict[str, Any] = None*) → int

Execute a command as admin but ensure that no stdout will be printed on the console

Parameters

- **commands** – the command to execute. They will be exeucte in the same context
- **cwd** – current working directory where the command is executed
- **env** – a dictionary representing the key-values of the environment variables

Returns pid of running process

execute_admin_return_stdout(*commands: Union[str, List[Union[str, List[str]]]]*, *cwd: str = None*, *env: Dict[str, Any] = None*, *check_exit_code: bool = True*, *timeout: int = None*) → Tuple[int, str, str]

Execute a command as an admin. We won't show the stdout on pmakeup console but we will capture it and returned it

Parameters

- **commands** – the command to execute. They will be execute in the same context
- **cwd** – current working directory where the command is executed
- **env** – a dictionary representing the key-values of the environment variables
- **check_exit_code** – if true, we will generate an exception if the exit code is different than 0
- **timeout** – if positive, we will give up waiting for the command after the amount of seconds

Returns triple. The first element is the error code, the second is the stdout (if captured), the third is stderr

execute_admin_stdout_on_screen(*commands: Union[str, List[Union[str, List[str]]]]*, *cwd: str = None*, *env: Dict[str, Any] = None*, *check_exit_code: bool = True*, *timeout: int = None*) → int

Execute a command as an admin. We won't capture the stdout but we will show it on pmakeup console

Parameters

- **commands** – the command to execute. They will be execute in the same context
- **cwd** – current working directory where the command is executed
- **env** – a dictionary representing the key-values of the environment variables
- **check_exit_code** – if true, we will generate an exception if the exit code is different than 0
- **timeout** – if positive, we will give up waiting for the command after the amount of seconds

Returns triple. The first element is the error code, the second is the stdout (if captured), the third is stderr

execute_admin_with_password_and_run_in_background(*commands: Union[str, List[Union[str, List[str]]]]*, *password: str*, *cwd: str = None*, *env: Dict[str, Any] = None*) → int

Execute a command as admin but ensure that no stdout will be printed on the console

Parameters

- **commands** – the command to execute. They will be exeucte in the same context
- **password** – password of the user to invoke the program as an admin
- **cwd** – current working directory where the command is executed
- **env** – a dictionary representing the key-values of the environment variables

Returns triple. The first element is the error code, the second is the stdout (if captured), the third is stderr

execute_admin_with_password_fire_and_forget(*commands: Union[str, List[Union[str, List[str]]]]*, *password: str*, *cwd: str = None*, *env: Dict[str, Any] = None*, *check_exit_code: bool = True*, *timeout: int = None*) → int

Execute a command as admin by providing the admin password. **THIS IS INCREDIBLE UNSAFE!!!!!!!!!!!!**. Please, I beg you, do **NOT** use this if you need any level of security!!!!! This will make the password visible on top, on the history, everywhere on your system. Please use it only if you need to execute a command on your local machine.

Parameters

- **commands** – the command to execute. They will be executed in the same context
- **cwd** – current working directory where the command is executed
- **env** – a dictionary representing the key-values of the environment variables
- **check_exit_code** – if true, we will generate an exception if the exit code is different than 0
- **timeout** – if positive, we will give up waiting for the command after the amount of seconds
- **password** – **[UNSAFE!!!!]** If you **really** need, you might want to run a command as an admin only on your laptop, and you want a really quick and dirty way to execute it, like as in the shell. Do **not** use this in production code, since the password will be 'printed in clear basically everywhere! (e.g., history, system monitor, probably in a file as well)

execute_admin_with_password_return_stdout(*commands: Union[str, List[Union[str, List[str]]]]*, *password: str*, *cwd: str = None*, *env: Dict[str, Any] = None*, *check_exit_code: bool = True*, *timeout: int = None*) → Tuple[int, str, str]

Execute a command as an admin. We won't show the stdout on pmakeup console but we will capture it and returned it

Parameters

- **commands** – the command to execute. They will be execute in the same context
- **password** – **[UNSAFE!!!!]** If you **really** need, you might want to run a command as an admin only on your laptop, and you want a really quick and dirty way to execute it, like as in the shell. Do **not** use this in production code, since the password will be 'printed in clear basically everywhere! (e.g., history, system monitor, probably in a file as well)
- **cwd** – current working directory where the command is executed
- **env** – a dictionary representing the key-values of the environment variables
- **check_exit_code** – if true, we will generate an exception if the exit code is different than 0
- **timeout** – if positive, we will give up waiting for the command after the amount of seconds

Returns triple. The first element is the error code, the second is the stdout (if captured), the third is stderr

execute_admin_with_password_stdout_on_screen(*commands: Union[str, List[Union[str, List[str]]]]*, *password: str*, *cwd: str = None*, *env: Dict[str, Any] = None*, *check_exit_code: bool = True*, *timeout: int = None*) → int

Execute a command as an admin. We won't capture the stdout but we will show it on pmakeup console

Parameters

- **commands** – the command to execute. They will be execute in the same context
- **password** – **[UNSAFE!!!!]** If you **really** need, you might want to run a command as an admin only on your laptop, and you want a really quick and dirty way to execute it, like as in the shell. Do **not** use this in production code, since the password will be 'printed in clear basically everywhere! (e.g., history, system monitor, probably in a file as well)
- **cwd** – current working directory where the command is executed
- **env** – a dictionary representing the key-values of the environment variables
- **check_exit_code** – if true, we will generate an exception if the exit code is different than 0
- **timeout** – if positive, we will give up waiting for the command after the amount of seconds

Returns triple. The first element is the error code, the second is the stdout (if captured), the third is stderr

execute_and_forget(*commands: Union[str, List[Union[str, List[str]]]]*, *cwd: str = None*, *env: Dict[str, str] = None*, *check_exit_code: bool = True*, *timeout: int = None*) → int

Execute a command but ensure that no stdout will be printed on the console

Parameters

- **commands** – the command to execute. They will be exeucte in the same context
- **cwd** – current working directory where the command is executed
- **env** – a dictionary representing the key-values of the environment variables
- **check_exit_code** – if true, we will generate an exception if the exit code is different than 0
- **timeout** – if positive, we will give up waiting for the command after the amount of seconds

Returns triple. The first element is the error code, the second is the stdout (if captured), the third is stderr

execute_and_run_in_background(*commands: Union[str, List[Union[str, List[str]]]]*, *cwd: str = None*, *env: Dict[str, str] = None*) → int

Execute a command but ensure that no stdout will be printed on the console

Parameters

- **commands** – the command to execute. They will be exeucte in the same context
- **cwd** – current working directory where the command is executed
- **env** – a dictionary representing the key-values of the environment variables

Returns pid of running process

execute_return_stdout(*commands: Union[str, List[Union[str, List[str]]]]*, *cwd: str = None*, *env: Dict[str, Any] = None*, *check_exit_code: bool = True*, *timeout: int = None*) → Tuple[int, str, str]

Execute a command. We won't show the stdout on pmakeup console but we will capture it and returned it

Parameters

- **commands** – the command to execute. They will be exeucte in the same context
- **cwd** – current working directory where the command is executed
- **env** – a dictionary representing the key-values of the environment variables
- **check_exit_code** – if true, we will generate an exception if the exit code is different than 0
- **timeout** – if positive, we will give up waiting for the command after the amount of seconds

Returns triple. The first element is the error code, the second is the stdout (if captured), the third is stderr

execute_stdout_on_screen(*commands: Union[str, List[Union[str, List[str]]]]*, *cwd: str = None*, *env: Dict[str, Any] = None*, *check_exit_code: bool = True*, *timeout: int = None*) → int

Execute a command. We won't capture the stdout but we will show it on pmakeup console

Parameters

- **commands** – the command to execute. They will be exeucte in the same context
- **cwd** – current working directory where the command is executed
- **env** – a dictionary representing the key-values of the environment variables
- **check_exit_code** – if true, we will generate an exception if the exit code is different than 0
- **timeout** – if positive, we will give up waiting for the command after the amount of seconds

Returns triple. The first element is the error code, the second is the stdout (if captured), the third is stderr

get_program_path() → Iterable[str]
List of paths in PATH environment variable

Returns collections of path

is_program_installed(*program_name: str*) → bool
Check if a program is reachable via commandline. We will look **only** in the PATH environment variable. If you want to look in other parts as well, conside rusing

Parameters **program_name** – the name of the program (e.g., dot)

Returns true if there is a program accessible to the PATH with the given name, false otherwise

CHAPTER

FOUR

PMAKEUP CACHE

class pmakeup.**IPMakeupCache**

abstract get_name() → str
human friendly name of the cache

abstract get_variable_in_cache(*name: str*) → Any
Obtain the variable value from the cache

Parameters **name** – the name of the varaible to obtain

Returns the variable obtained

abstract has_variable_in_cache(*name: str*) → bool
Check if the variable is present in the cache

Parameters **name** – the name of the variable to check

Returns true if the variable is present inthe cache, false otherwise

abstract is_cache_present() → bool
Check if the pmakeup cache is present

abstract is_empty() → bool
Check if there is at least one variable in cache

Returns true iff there is no variables in cache

abstract reset()
Completely empty the pmakeupfile. After the operation, the cache is present, but it is empty. It is required to persistently update the cache in this method

abstract set_variable_in_cache(*name: str*, *value: Any*, *overwrites_is_exists: bool = True*)
Set a variable in the cache.

Parameters

- **name** – name of the variable to add

- **value** – value to store
- **overwrites_is_exists** – if true, we will overwrite any previous variable in the cache

abstract update_cache()
Store the cache persistently

abstract variable_names() → Iterable[str]
Set of variable names available in the cache. They are only at top level

class pmakeup.**JsonPMakeupCache**(*file_path: str*)

get_name() → str
human friendly name of the cache

get_variable_in_cache(*name: str*) → Any
Obtain the variable value from the cache

Parameters **name** – the name of the varaible to obtain

Returns the variable obtained

has_variable_in_cache(*name: str*) → bool
Check if the variable is present in the cache

Parameters **name** – the name of the variable to check

Returns true if the variable is present inthe cache, false otherwise

is_cache_present() → bool
Check if the pmakeup cache is present

is_empty() → bool
Check if there is at least one variable in cache

Returns true iff there is no variables in cache

reset()
Completely empty the pmakeupfile. After the operation, the cache is present, but it is empty. It is required to persistently update the cache in this method

set_variable_in_cache(*name: str*, *value: Any*, *overwrites_is_exists: bool = True*)
Set a variable in the cache.

Parameters

- **name** – name of the variable to add
- **value** – value to store
- **overwrites_is_exists** – if true, we will overwrite any previous variable in the cache

update_cache()
Store the cache persistently

variable_names() → Iterable[str]
Set of variable names available in the cache. They are only at top level

CHAPTER

FIVE

CREATE A NEW PLUGIN

This is a tutorial to create a *pmakeup* plugin.

For an example of a plugin, you can view the *archive-pmakeup-plugin*, available *here <https://github.com/Koldar/pmakeup/tree/main/plugins/archive-pmakeup-plugin>*.

5.1 Determine the name of the plugin

pmakeup can automatically load plugins only if they are named in either one of the following pattern:

- r"^pmakeup-plugin(s)?-.+";
- r".+-pmakeup-plugin(s)?$";

At the beginning of the run of *pmakeup*, the software will first scan all the installed packages. When it finds a plugin with the specified name, it will further explores it. So, if you want to develop a plugin is **required to follow that patterns**.

5.2 Structure of a *pmakeup* plugin

Generally speaking, a pmakeup plugin should have the specified structure:

```
myfoobar-pmakeup-plugin/ (<-- this is just the plugin root folder)
    myfoobar_pmakeup_plugin/
        __init__.py
        MyFoobarPMakeupPlugin.py
        version.py
    tests/
        test.py
    README.md
    LICENSE.md
```

(continues on next page)

(continued from previous page)

```
    requirements.txt
    setup.py
```

You can also use `pmakeup` to automatically build and deploy your plugin on pypi, but this is another story. The example here specifies a `setuptools` installation way. `pmakeup` relies on egg-infos. We first look at the file `top_level.txt` in order to fetch the main package name. Then we gain access to such a package and read the `__init__.py`. Finally, we scan whatever `__init__.py` has loaded. If it finds a class which derives from `pm.AbstractPMakeupPlugin` it is autoamtically added in the `pmakeup` graph.

```
for apackage in map(lambda p: p, pkg_resources.working_set):
    package: "EggInfoDistribution" = apackage

    if not is_package_name_compliant_with_pmakeup_plugin_name(package.
↪project_name):
        continue

    # get top level file
    top_level_file = os.path.join(package.egg_info, "top_level.txt")
    main_package = read_top_level_file(top_level_file)
    module_path = os.path.join(package.location, main_package, "__init_
↪_.py")
    module = import_module(module_path, main_package)

    # fetch plugins
    for candidate_classname in dir(module):
        candidate_class = getattr(module_instance, candidate_classname)
        if not inspect.isclass(candidate_class):
            continue
        if issubclass(candidate_class, pm.AbstractPmakeupPlugin):
            result.append(candidate_class)
```

Now let's see what the files should contain.

5.3 version.py

This file is easy, it is the version of the package. It should contain one line with a variable set to a semantic version 2 compliant string:

```
VERSION = "1.0.4"
```

5.4 __init__.py

This file is very easy as well. It should import all the `pmakeup` plugin classes that you want to export to `pmakeup`. For instance, we will export just one plugin:

```
from myfoobar_pmakeup_plugin.MyFoobarPMakeupPlugin import␣
↪MyFoobarPMakeupPlugin
```

5.5 MyFoobarPMakeupPlugin.py

This file should contain a class that implements `pm.AbstractPMakeupPlugin`:

```
import pmakeup as pm

class MyFoobarPMakeupPlugin(pm.AbstractPMakeupPlugin):

    def _setup_plugin(self):
        pass

    def _teardown_plugin(self):
        pass

    def _get_dependencies(self) -> Iterable[type]:
        return []

    @pm.register_command.add("really_important")
    def say_hello(self, name: str) -> bool:
        """
        Say hello to everyone
        """

        self.logs.echo(f"Hello {name}!")
        return True
```

If you don't need that another plugin `_setup_plugin` method is called before this one, you can leave `_get_dependencies` to `[]`. setup and teardown methods are called whenever the plugin is initialized and finalized.

Any function that you want to call in a `pmakeup` script needs to be decorated with `@pm.register_command.add` decorator: the string can be whatever you want, it is used only for grouping the functions together.

If you need to gain access to other plugins, you can use `self.get_plugin(<plugin_name>)` to gain access to the corresponding plugin instance. `pmakeup` automatically loads some **really** core plugnis and it provides a property in

`AbstractPMakeupPlugin`: for example `self.logs` is used to print something to the console.

5.6 setup.py

Just for completeness, this is the `setup.py` that I use to build a plugin:

```
import os
from typing import Iterable

import setuptools
from archive_pmakeup_plugin import version

PACKAGE_NAME = "archive-pmakeup-plugin"
PACKAGE_VERSION = version.VERSION
PACKAGE_DESCRIPTION = "A Pmakeup plugin for handling zip and unzip␣
↪operations"
PACKAGE_URL = "https://github.com/Koldar/pmakeup.git"
PACKAGE_PYTHON_COMPLIANCE = ">=3.6"
PACKAGE_CLASSIFIERS = [
    "Programming Language :: Python :: 3",
    "License :: OSI Approved :: MIT License",
    "Operating System :: OS Independent",
]
AUTHOR_NAME = "Massimo Bono"
AUTHOR_EMAIL = "massimobono1@gmail.com"

############################################################
# INTERNALS
############################################################

with open("README.md", "r", encoding="utf-8") as fh:
    long_description = fh.read()


def get_dependencies(domain: str = None) -> Iterable[str]:
    if domain is None:
        filename = "requirements.txt"
    else:
        filename = f"requirements-{domain}.txt"
    if os.path.exists(filename):
        with open(filename, "r", encoding="utf-8") as fh:
            dep = fh.readline()
            dep_name = dep.split("==")[0]
            yield dep_name + ">=" + dep.split("==")[1]
```

(continues on next page)

(continued from previous page)

```
setuptools.setup(
    name=PACKAGE_NAME,
    version=PACKAGE_VERSION,
    author=AUTHOR_NAME,
    author_email=AUTHOR_EMAIL,
    description=PACKAGE_DESCRIPTION,
    long_description=long_description,
    long_description_content_type="text/markdown",
    license_files="LICEN[SC]E*",
    url=PACKAGE_URL,
    packages=setuptools.find_packages(),
    classifiers=PACKAGE_CLASSIFIERS,
    install_requires=list(get_dependencies()),
    extras_require={
        "test": list(get_dependencies("test")),
        "doc": list(get_dependencies("doc")),
    },
    include_package_data=True,
    package_data={
        "": ["package_data/*.*"],
    },
    python_requires=PACKAGE_PYTHON_COMPLIANCE,
)
```

CHAPTER

SIX

CODE DOCUMENTATION

here you can find all the code documentation of the project

class pmakeup.**AbstractPmakeupPlugin**(*model: pmakeup.models.PMakeupModel.PMakeupModel*)

Bases: abc.ABC

classmethod autoregister()
Function to call from the __init__ file of the plugin that allows the module to automatically be registered. If you put it in the __init_ file, as soon as the plugin is imported in your pmakeup script, the plugin will immediately be loaded. If you don't put it in the __init__ file, the developer writing the pmakeup script will have to do it herself by explicitly calling require_pmakeup_plugins

property core
Gain access to the core plugin, which is well populated

property files
Gain access to the core plugin, which is well populated

get_cwd() → str

Returns the CWD the current commands operates in, as absolute payj

get_plugin(*plugin: Union[str, type]*) → pmakeup.plugins.AbstractPmakeupPlugin.AbstractPmakeupP
Get a plugin of a prticular type

Parameters **plugin** – type of the plugin to find or the plugin name

Returns instance of the given plugin. Raises an exception if not found

get_plugin_functions() → Iterable[Tuple[str, Callable]]
Yield all the functions registered by this plugin

get_plugin_name()
The name of the plugin. Useful to fetch plugin dynamically

get_plugins() → Iterable[pmakeup.plugins.AbstractPmakeupPlugin.AbstractPmakeupPlugin]
get all plugins registered up to this point

get_registry() → pmakeup.models.PMakeupRegistry.PMakeupRegistry
get the pmakeup registry, where all shared entities available for plugins are located

get_shared_variables() → pmakeup.models.AttrDict.AttrDict

get_variable(*name: str*) → Any
Ensure the user has passed a variable. If not, raises an exception

Parameters **name** – the variable name to check

Raises ***PMakeupException*** – if the variable is not found

get_variable_or_set_it(*name: str*, *otherwise: Any*) → Any
Ensure the user has passed a variable. If not, the default variable is stored in the variable sety

Parameters

- **name** – the variable name to check
- **otherwise** – the value the varible with name will have if the such a variable is not present

has_plugin(*plugin_type: type*) → bool
Check if a plugin has been loaded

property is_setupped
true if the function setup has already been called, false otherwise

property logs
Gain access to the core plugin, which is well populated

property operating_system
Gain access to the operating system plugin, which is well populated

property paths
Gain access to the core plugin, which is well populated

property platform
fetch the plugin repersenting the operating system on this machine

set_cwd(*value*)
set the CWD the current commands operates in :param value: new value of the CWD

set_variable(*name: str*, *value: Any*) → None
Set the variable in the current model. If the variable did not exist, we create one one. Otherwise, the value is overridden

Parameters

- **name** – name of the variable to programmatically set
- **value** – value to set

exception pmakeup.**AssertionPMakeupException**
Bases: pmakeup.exceptions.PMakeupException.PMakeupException

class pmakeup.**AttrDict**(*d*)
Bases: object

has_key(*item: str*) → bool

items() → Iterable[Tuple[int, Any]]

keys() → Iterable[str]

values() → Iterable[Any]

class pmakeup.**CorePMakeupPlugin**(*model: pmakeup.models.PMakeupModel.PMakeupModel*)
Bases: pmakeup.plugins.AbstractPmakeupPlugin.AbstractPmakeupPlugin

Contains all the commands available for the user in a PMakeupfile.py file

add_or_update_variable_in_cache(*name: str*, *supplier: Callable[], Any]*, *mapper: Callable[[Any], Any]*)
Add a new variable in the cache

Parameters

- **name** – the variable to set
- **supplier** – function used to generate the value fo the variable if the variable does not exist in the cache
- **mapper** – function used to generate the value fo the variable if the variable does exist in the cache. The input is the variable old value

clear_cache()
Clear the cache of pmakeup

ensure_condition(*condition: Callable[], bool]*, *message: str = ''*) → None
Perform a check. If the condition is **not** satisfied, we raise exception

Parameters

- **condition** – the condition to check. generate exception if the result is False
- **message** – the message to show if the exception needs to be generated

ensure_has_cli_variable(*name: str*) → None
Ensure the user has passed a variable via "–variable" CLI utils. If not, an exception is generated

Parameters **name** – the variable name to check

ensure_has_cli_variable_is_one_of(*name: str*, **allowed_values*) → None
Ensure that a variable has been passed from the command line and has a value among the one passed

Parameters

- **name** – variable name
- **allowed_values** – set of values we check against the variable calue

ensure_has_variable(*name: str*) → None
Ensure the user has passed a variable in the registry. If not, an exception is generated

Parameters **name** – the variable name to check

get_all_available_command_names() → Iterable[str]
Get all the commands you can execute right now

get_all_registered_plugins() → Iterable[str]
get all the registered pmakeup plugins at this moment

get_architecture() → int
check if the system is designed on a 32 or 64 bits

Returns either 32 or 64 bit

get_command_line_string() → str
Get the command line string from the user

Returns argv

get_home_folder() → str
Get the home fodler of the currently logged used

get_latest_path_with_architecture(*current_path: str*, *architecture: int*) → str
get the latest path on the system with the specified archietcture

Parameters

- **current_path** – nominal path name
- **architecture** – either 32 or 64

Returns the first path compliant with this path name

get_latest_version_in_folder(*folder: str = None*, *should_consider: Callable[[str], bool] = None*, *version_fetcher: Callable[[str], semantic_version.base.Version] = None*) → Tuple[semantic_version.base.Version, List[str]]
Scan the subfiles and subfolder of a given directory. We assume each file or folder

has a version withint it. Then fetches the latest version. This command is useful in dierctories where all releases of a given software are placed. if we need to fetch the latest one, this function is perfect for the task.

Parameters

- **folder** – the folder to consider. If unspecified, it is the current working directory
- **should_consider** – a function that allows you to determine if we need to consider or not a subfile/subfolder. The input isan absolute path. If no function is given, we accept all the sub files
- **version_fetcher** – a function that extract a version from the filename. If left unspecified, we will use ::semantic_version_2_only_core

Returns the latest version in the folder. The second element of the tuple is a collection of all the filenames that specify the latest version

get_pmakeupfile_dir() → str

The directory where the analyzed pmakeupfile is located

Returns absolute ptha of the directory of the path under analysis

get_pmakeupfile_dirpath() → str

Returns absolute path of the folder containing the main PMakeupfile path

get_pmakeupfile_path() → str

Returns absolute path of the main PMakeupfile path

get_starting_cwd() → str

Returns absolute path of where you have called pmakeup

get_variable_in_cache(*name: str*) → Any

Get the variable from the cache. if the variable does not exist, an error is generated

Parameters **name** – name of the variable to check

Returns the value associated to such a variable

get_variable_in_cache_or(*name: str*, *default: Any*) → Any

Get the variable value from the cache or get a default value if it does not exist

Parameters

- **name** – name of the variable to fetch
- **default** – if the variable does not exist in the cache, the value to retturn from this function

Returns the variable value

get_variable_in_cache_or_fail(*name: str*) → Any
Get the variable value from the cache or raise an error if it does not exist

Parameters **name** – name of the variable to fetch

Returns the variable value

has_variable_in_cache(*name: str*) → bool
Check if a variable is in the pmakeup cache

Parameters **name** – name of the variable to check

Returns true if a varaible with such a name is present in the cache, false otherwise

include_file(**file: str*) → None
Replace the include directive with the content fo the included file. Fails if there is no such path

Parameters **file** – the external file to include in the script

include_string(*string: str*) → None
Include and execute the code within the given string

Parameters **string** – the commands to execute

is_process_running(*program_name: str*) → bool
Check if a program with the given name is currently running

Parameters **program_name** – the program we need to check

Returns true if we are running such a program, false otheriwse

kill_process_by_name(*program_name: str*, *ignore_if_process_does_not_exists: bool = True*)
Kill a program

Parameters

- **program_name** – name fo the program that is running on the system
- **ignore_if_process_does_not_exists** – if the proces does not exist and thsi parameter is true, this function will **not** throw exception

kill_process_by_pid(*pid: int*, *ignore_if_process_does_not_exists: bool = True*)
Kill a program

Parameters

- **pid** – pid of the program that is running on the system

- **ignore_if_process_does_not_exists** – if the proces does not exist and thsi parameter is true, this function will **not** throw exception

load_cache()
Load all the variables present in cache into the available variables

log_command(*message: str*)
reserved. Useful to log the action performed by the user

Parameters **message** – message to log

on_linux() → bool
Check if we are running on linux

Returns true if we are running on linux

on_windows() → bool
Check if we are running on windows

Returns true if we are running on windows

path_wrt_pmakeupfile(**folder: str*) → str
Compute path relative to the file where PMakeupfile is located

Parameters **folder** – other sections of the path

Returns path relative to the absolute path of where PMakeupfile is located

path_wrt_starting_cwd(**folder: str*) → str
Compute path relative to the starting cwd

Parameters **folder** – other sections of the path

Returns path relative to the absolute path of where you have called pmakeup

quasi_semantic_version_2_only_core(*filename: str*) → semantic_version.base.Version
A function that can be used within ::get_latest_version_in_folder. It accepts values like "1.0.0", but also "1.0" and "1"

Parameters **filename** – the absolute path of a file that contains a version

Returns the version

read_variables_from_properties(*file: str*, *encoding: str = 'utf-8'*) → None
Read a set of easy variables from a property file. All the read variables will be available in the "variables" value. If some variable name preexists, it will not be overriden :see: https://docs.oracle.com/cd/E23095_01/Platform.93/ATGProgGuide/html/s0204propertiesfileformat01.html

Parameters

- **file** – the file to read

- **encoding** – encoding of the file. If left missing, we will use utf-8

require_pmakeup_plugins (**pmakeup_plugin_names: str*)
Tells pmakeup that, in order to run the script, you required a sequence of pmakeup plugins correctly installed (the version does not matter)

Pmakeup will then arrange itself in installing dependencies and the correct order of the plugins

Parameters **pmakeup_plugin_names** – the plugins that are requierd to be present in order for the script to work. Dependencies are automatically added

require_pmakeup_version (*lowerbound: str*) → None
Check if the current version of pmakeup is greater or equal than the given one. If the current version of pmakeup is not compliant with this constraint, an error is generated

Parameters **lowerbound** – the minimum version this script is compliant with

semantic_version_2_only_core (*filename: str*) → semantic_version.base.Version
A function that can be used within ::get_latest_version_in_folder

Parameters **filename** – the absolute path of a file that contains a version

Returns the version

set_variable_in_cache (*name: str*, *value: Any*, *overwrite_if_exists: bool = True*)
Set a variable inside the program cache. Setting variable in cache allows pmakeup to store information between several runs of pmakeup.

How pmakeup stores the information is implementation dependent and it should not be relied upon

Parameters

- **name** – name of the variable to store
- **value** – object to store
- **overwrite_if_exists** – if true, if the cache already contain a variable with the same name, such a varaible will be replaced with the new one

vars () → pmakeup.models.AttrDict.AttrDict
Get a dictioanry containing all the variables setup up to this point. You can use thi dictionary to gain access to a variable in a more pythonic way (e.g., vars.foo rather than get_variable("foo")

Raises ***PMakeupException*** – if the variable is not found

class pmakeup.**FilesPMakeupPlugin**(*model: pmakeup.models.PMakeupModel.PMakeupModel*)

Bases: pmakeup.plugins.AbstractPmakeupPlugin.AbstractPmakeupPlugin

allow_file_to_be_executed_by_anyone(*file: str*)

Allow the file to be executed by anyone. On a linux system it should be equal to "chmod o+x"

Parameters **file** – the file whose permission needs to be changed

append_string_at_end_of_file(*name: str*, *content: Any*, *encoding: str = 'utf-8'*) → None

Append a string at the end of the file. carriage return is automatically added

Parameters

- **name** – filename
- **content** – string to append
- **encoding** – encoding of the file. If missing, "utf-8" is used

append_strings_at_end_of_file(*name: str*, *content: Iterable[Any]*, *encoding: str = 'utf-8'*) → None

Append a string at the end of the file. carriage return is automatically added

Parameters

- **name** – filename
- **content** – string to append
- **encoding** – encoding of the file. If missing, "utf-8" is used

copy_file(*src: str*, *dst: str*, *create_dirs: bool = True*)

Copy a single file from a position to another one. If the destination folder hierarchy does not exist, we will create it

Parameters

- **src** – file to copy
- **dst** – destination where the file will be copied to. If a file, we will copy the src file into another file with different name. If a directory, we will copy the specified file into the dirctory dst (without altering the filename)
- **create_dirs** – if true, we will create the directories of dst if non existent

copy_files_that_basename(*src: str*, *dst: str*, *regex: str*)

Copy the files located (directly or indirctly) in src into dst. We will copy only the files whose basename (e.g. foo.txt is the basename of /opt/foo/bar/foo.txt). We will copy the directories where a file is located as well matches the given regex

Parameters

- **src** – folder where we will find files to copy
- **dst** – destination of the files
- **regex** – regex that determines wether or not a file is copies

Returns

copy_folder_content (*folder: str*, *destination: str*)
Copy all the content of "folder" into the folder "destination"

Parameters

- **folder** – folder to copy files from
- **destination** – folder where the contents will be copied into

copy_tree (*src: str*, *dst: str*)
Copy a whole directory tree or a single file. If you specifies a file rather than a directory, the function behaves like :see copy_file

Parameters

- **src** – the folder or the file to copy.
- **dst** – the destination where the copied folder will be positioned

create_empty_directory (*name: str*) → str
Create an empty directory in the CWD (if the path is relative)

:param name:the name of the driectory to create :return: the full path of the directory just created

create_empty_file (*name: str*, *encoding: str = 'utf-8'*)
Create an empty file. if the file is relative, it is relative to the CWD

Parameters

- **name** – file name to create
- **encoding** – encoding of the file. If unspecified, it is utf-8

find_directory (*root_folder: str*, *folder: str*) → Iterable[str]
Find all the directories with the given name

Parameters

- **root_folder** – fodler where we need to look int
- **folder** – name of the folder we need to fetch

Returns list of files with thwe given filename

find_directory_with_filename_compliant_with_regex(*root_folder: str*, *folder_regex: str*) → Iterable[str]

Find all the directories with the given name

Parameters

- **root_folder** – fodler where we need to look int
- **folder_regex** – regex the folder name should be compliant with

Returns list of files with thwe given filename

find_directory_with_fullpath_compliant_with_regex(*root_folder: str*, *folder_regex: str*) → Iterable[str]

Find all the directories with the given name

Parameters

- **root_folder** – folder where we need to look int
- **folder_regex** – regex the folder name should be compliant with

Returns list of files with thwe given filename

find_executable_in_program_directories(*program_name: str*, *fail_if_program_is_not_found: bool = False*) → Optional[str]

Find a program ouside the path as well. Paths is still considered

Parameters

- **program_name** – name of the program to look for
- **fail_if_program_is_not_found** – if true, we will raise an exception if the program is not found

Returns first absolute path of the program found. None if we did not find the program

find_file(*root_folder: str*, *filename: str*) → Iterable[str]

Find all the files with the given filename (extension included)

Parameters

- **root_folder** – fodler where we need to look int
- **filename** – filename we need to fetch

Returns list of files with thwe given filename

find_file_in_roots_st(*root_folders: str*, *match: Callable[[str, str, str], bool]*) → Iterable[str]

Find all the files matchign the given function

Parameters

- **root_folders** – folders where we need to look int
- **match** – a function that defines if you want to include the file into the output. The first parameter is the folder containing the given file. The second parameter is the involved file. The third is the absolute path of the involved path

Returns list of files compliant with the given function

find_file_st(*root_folder: str*, *match: Callable[[str, str, str], bool]*) → Iterable[str]

Find all the files matchign the given function

Parameters

- **root_folder** – folder where we need to look int
- **match** – a function that defines if you want to include the file into the output. The first parameter is the folder containing the given file. The second parameter is the involved file. The third is the absolute path of the involved path

Returns list of files compliant with the given function

find_file_with_filename_compliant_with_regex(*root_folder: str*, *filename_regex: str*) → Iterable[str]

Find all the files containign (search) the given regex

Parameters

- **root_folder** – folder where we need to look int
- **filename_regex** – the regex any filename should be compliant

Returns list of files with thwe given filename

find_file_with_fullpath_compliant_with_regex(*root_folder: str*, *filename_regex: str*) → Iterable[str]

Find all the files containing (search) the given regex

Parameters

- **root_folder** – folder where we need to look int
- **filename_regex** – the regex any filename should be compliant

Returns list of files with the given filename

find_first_file_in_roots_st(*root_folders: str*, *match: Callable[[str, str, str], bool]*) → Optional[str]

Find the first file matching the given function

Parameters

- **root_folders** – folders where we need to look int
- **match** – a function that defines if you want to include the file into the output. The first parameter is the folder containing the given file. The second parameter is the involved file. The third is the absolute path of the involved path

Returns file compliant with the given function or None

find_first_file_in_roots_st_or_fail(*root_folders: str*, *match: Callable[[str, str, str], bool]*) → str

Find the first file matching the given function. If no such file exists, generates an exception

Parameters

- **root_folders** – folders where we need to look int
- **match** – a function that defines if you want to include the file into the output. The first parameter is the folder containing the given file. The second parameter is the involved file. The third is the absolute path of the involved path

Returns file compliant with the given function or None

find_first_file_st(*root_folder: str*, *match: Callable[[str, str, str], bool]*) → Optional[str]

Find the first file matching the given function

Parameters

- **root_folder** – folder where we need to look int
- **match** – a function that defines if you want to include the file into the output. The first parameter is the folder containing the given file. The second parameter is the involved file. The third is the absolute path of the involved path

Returns file compliant with the given function or None

find_first_file_st_or_fail(*root_folder: str*, *match: Callable[[str, str, str], bool]*) → str

Find the first file matching the given function. If no such file exists, generates an exception

Parameters

- **root_folder** – folder where we need to look int
- **match** – a function that defines if you want to include the file into the output. The first parameter is the folder containing the given file. The second parameter is the involved file. The third is the absolute path of the involved path

Returns file compliant with the given function or None

find_folder_st(*root_folder: str*, *match: Callable[[str, str, str], bool]*) → Iterable[str]

Find all the folder matching a given function

Parameters

- **root_folder** – folder where we need to look int
- **match** – a function that defines if you want to include the folder into the output. The first parameter is the folder containing the given folder. The second parameter is the involved folder. The third is the absolute path of the involved path

Returns list of folders compliant with the given function

find_regex_match_in_file(*pattern: str*, **p: str*, *encoding: str = 'utf8'*, *flags: Union[int, re.RegexFlag] = 0*) → Optional[re.Match]

FInd the first regex pattern in the file

If you used named capturing in the pattern, you can gain access via result.group("name")

Parameters

- **pattern** – regex pattern to consider
- **p** – file to consider
- **encoding** – encoding of the file to search. Defaults to utf8
- **flags** – flags of the regex to build. Passed as-is

Returns a regex match representing the first occurence. If None we could not find anything

get_file_size(**f: str*) → int

Get the filesize of a given file. If the file is a directory, return the cumulative size of all

the files in it

Parameters **f** – the path of the file to consider

Returns number of bytes

is_directory(**p: str*) → bool
Check if the given path is a directory

Parameters **p** – paths to check

Returns true if the concatenated version of p is a directory. False otherwise

is_directory_empty(*name: str*) → bool
Check if a directory exists and is empty

Parameters **name** – folder to check

Returns true if the folder exists and is empty, false otherwise

is_directory_exists(*name: str*) → bool
Check if a directory exists.

Parameters **name** – folder to check

Returns true if the folder exists, false otherwise

is_file(**p: str*) → bool
Check if the given path represents a file or a directory

Parameters **p** – paths to check

Returns true if the concatenated version of p is a file. False otherwise

is_file_empty(*name: str*) → bool
Checks if a file exists. If exists, check if it empty as well.

Parameters **name** – file to check

Returns true if the file exists **and** has no bytes; false otherwise

is_file_exists(*name: str*) → bool
Check if a file exists

Parameters **name** – file whose existence we need to assert

Returns true if the file exists, false otherwise

is_file_non_empty(**name: str*) → bool
Checks if a file exists. If exists, check if it is not empty as well.

Parameters **name** – file to check

Returns true if the file exists **and** has at least one byte; false otherwise

ls(*folder: str = None*, *generate_absolute_path: bool = False*) → Iterable[str]
Show the list of all the files in the given directory

Parameters

- **folder** – folder to scan. default to CWD
- **generate_absolute_path** – if true, we will generate in the outptu the absolute path of the subfolders. Otherwise we will return only the

Returns iterable of all the files in the given directory

ls_directories_recursive(*folder: str*) → Iterable[str]
Show the list of all the directories in the given folder

Parameters **folder** – folder to scan (default to cwd)

Returns list of absolute filename representing the stored directories

ls_only_directories(*folder: str = None*, *generate_absolute_path: bool = False*) → Iterable[str]
Show the list of all the directories in the given directory

Parameters

- **folder** – folder to scan. If missing, default to CWD
- **generate_absolute_path** – if true, we will generate in the outptu the absolute path of the subfolders. Otherwise we will return only the names.

Returns a list of absolute paths representing the subdirectories inside `folder`

ls_only_files(*folder: str = None*, *generate_absolute_path: bool = False*) → Iterable[str]
Show the list of all the files (but not directories) in the given directory

Parameters

- **folder** – folder to scan. default to CWD
- **generate_absolute_path** – if true, we will generate in the outptu the absolute path of the subfolders. Otherwise we will return only the

Returns

ls_recursive(*folder: str = None*) → Iterable[str]
Show the list of all the files in the given folder

Parameters **folder** – folder to scan (default to cwd)

Returns list of absolute filename representing the stored files

make_directories(**folder: str*) → None
Create all the needed directories for the given path. Note that if you inject the path

temp/foo/hello.txt (you can see hello.txt shoudl be a file) the function will generate hello.txt as a **directory**!

Parameters **folder** – folders to create

move_file(*src: str*, *dst: str*)
Move a single file from a location to another one

Parameters

- **src** – the file to move
- **dst** – the path where the file will be moved to

move_tree(*src: str*, *dst: str*)
Move an entire directory tree from one position to another one

Parameters

- **src** – path of the directory to move
- **dst** – path of the directory that we will create

read_file_content(*name: str*, *encoding: str = 'utf-8'*, *trim_newlines: bool = True*) → str
Read the whole content of the file in a single string

Parameters

- **name** – name of the file to load
- **encoding** – the encoding of the file. If unspecified, it is utf-8
- **trim_newlines** – if true, we will trim the newlines, spaces and tabs at the beginning and at the end of the file

Returns string repersenting the content of the file

read_lines(*name: str*, *encoding: str = 'utf-8'*) → Iterable[str]
Read the content of a file and yields as many item as there are lines in the file. Strip from the line ending new lines. Does not consider empty lines

Parameters

- **name** – name of the file
- **encoding** – encoding of the file. If unspecified, it is utf-8

Returns iterable containing the lines of the file

remove_file(*name: str*, *ignore_if_not_exists: bool = True*) → bool
Remove a file. If the cannot be removed (for some reason), ignore_if_not_exists determines if somethign goes wrong

Parameters

- **name** – file to delete
- **ignore_if_not_exists** – if true, we won't raise exception if the file does not exists or cannot be removed

Returns true if we have removed the file, false otherwise

remove_files_that_basename(*src: str*, *regex: str*)

Remove the files located (directly or indirectly) in src. We will copy only the files whose basename (e.g. foo.txt is the basename of /opt/foo/bar/foo.txt). We will copy the directories where a file is located as well matches the given regex

Parameters

- **src** – folder where we will find files to copy
- **regex** – regex that determines wether or not a file is copies

Returns

remove_last_n_line_from_file(*name: str*, *n: int = 1*, *consider_empty_line: bool = False*, *encoding: str = 'utf-8'*) → List[str]

Read the content of a file and remove the last n lines from the file involved. Then, rewrites the whole file

Parameters

- **name** – file involved. If relative, it is relative to ::cwd()
- **n** – the number of lines to remove at the end.
- **consider_empty_line** – if True, we consider empty lines as well.
- **encoding** – the encoding used to rewrite file

Returns the lines just removed

remove_string_in_file(*name: str*, *substring: str*, *count: int = - 1*, *encoding: str = 'utf-8'*)

Remove some (or all) the occurences of a given substring in a file

Parameters

- **name** – path of the file to handle
- **substring** – substring to replace
- **count** – the number of occurences to remove. -1 if you want to remove all occurences
- **encoding** – encoding used for reading the file

remove_tree(**folder: str*, *ignore_if_not_exists: bool = True*) → None

Remove a dirctory tree

Parameters

- **folder** – path to the directory to remove
- **ignore_if_not_exists** – if the directory does not exists, we do nothing if htis field is true

replace_regex_in_file(*name: str*, *regex: str*, *replacement: str*, *count: int = -1*, *encoding: str = 'utf-8'*)

Replace some (or all) the occurences of a given regex in a file.

If you want to use named capturing group, you can do so! For instance,

replace_regex_in_file(file_path, '(?P<word>w+)', '(?P=word)aa') 'spring' will be replaced with 'springaa'

It may not work, so you can use the following syntax to achieve the same: replace_regex_in_file(file_path, '(?P<word>w+)', r'g<word>aa') 'spring' will be replaced with 'springaa'

Parameters

- **name** – path of the file to handle
- **regex** – regex to replace
- **replacement** – string that will replace *substring*
- **count** – the number of occurences to replace. -1 if you want to replace all occurences
- **encoding** – encoding used for reading the file

See https://docs.python.org/3/howto/regex.html

replace_string_in_file(*name: str*, *substring: str*, *replacement: str*, *count: int = -1*, *encoding: str = 'utf-8'*)

Replace some (or all) the occurences of a given substring in a file

Parameters

- **name** – path of the file to handle
- **substring** – substring to replace
- **replacement** – string that will replace *substring*
- **count** – the number of occurences to replace. -1 if you want to replace all occurences
- **encoding** – encoding used for reading the file

write_file(*name: str*, *content: Any*, *encoding: str = 'utf-8'*, *overwrite: bool = False*, *add_newline: bool = True*)

Write into a file with the specified content. if overwrite is unset, we will do nothing if the file already exists

Parameters

- **name** – name of the file to create
- **content** – content of the file to create.
- **encoding** – encoding fo the file to create. utf-8 by default
- **overwrite** – if true, we will overwrite the file
- **add_newline** – if true, we will add a new line at the end of the content

write_lines(*name: str*, *content: Iterable[Any]*, *encoding: str = 'utf-8'*, *overwrite: bool = False*)

Write severla lines into a file. if overwrite is unset, we will do nothing if the file already exists

Parameters

- **name** – name of the file to create
- **content** – lines of the file to create. We will append a new ine at the end of each line
- **encoding** – encoding fo the file to create. utf-8 by default
- **overwrite** – if true, we will overwrite the file

class pmakeup.**IOSSystem**

Bases: abc.ABC

create_temp_directory_with(*directory_prefix: str*) → Any

Create a temporary directory on the file system where to put temporary files

Parameters **directory_prefix** – a prefix to be put before the temporary folder

Returns a value which can be the input of a "with" statement. The folder will be automatically removed at the end of the with. The value returned is actually the absolute path of the temp directory

create_temp_file(*directory: str*, *file_prefix: Optional[str] = None*, *file_suffix: Optional[str] = None*, *readable_for_all: bool = False*, *executable_for_owner: bool = False*, *executable_for_all: bool = False*) → str

Creates the file Like ::create_temp_file_with, but the file needs to be manually removed

Parameters

- **directory** – the directory where to put the file
- **file_prefix** – a string that will be put at the beginning of the filename

- **file_suffix** – a string that will be put at the end of the filename
- **readable_for_all** – if True, the file can be read by anyone
- **executable_for_owner** – if True, the file can be executed by the owner
- **executable_for_all** – if True, anyone can execute the file

Returns the absolute path of the temp file

create_temp_file_with(*directory: str*, *file_prefix: Optional[str] = None*, *file_suffix: Optional[str] = None*, *encoding: Optional[str] = None*, *mode: Optional[str] = None*) → Any

Create a temporary file on the file system. The return value of this function is something you can give to the "with" statement. The file will be automatically remove at the end of the with. You can access the file absolute path via the field "name" of the return value

Parameters

- **directory** – the directory where to put the file
- **file_prefix** – a string that will be put at the beginning of the filename
- **file_suffix** – a string that will be put at the end of the filename
- **encoding** – encoding used to open the file
- **mode** – the mode used to open the file. E.g., "w", "r", "w+". See open for further information

Returns a return value that can be used as input of with statement

abstract fetch_interesting_paths(*model: pmakeup.models.PMakeupModel.PMakeupModel*) → Dict[str, List[pmakeup.platforms.InterestingPath.InterestingPath]]

Fetch all the interesting paths relative to a operating system. Highly dependent on the operating system. Each path has associated different actual paths, since a single

Parameters **model** – model of the pmakeup

Returns

fetch_latest_interesting_paths(*interesting_paths: Dict[str, List[pmakeup.platforms.InterestingPath.InterestingPath]]*, *model: pmakeup.models.PMakeupModel.PMakeupModel*) → Dict[str, pmakeup.platforms.InterestingPath.InterestingPath]

Fetch for every path only one path which is interesting in your case. For instance, there

may be multiple internet explorer executables, but you need to use only a specific one

abstract find_executable_in_program_directories(*program_name: str*) → Optional[str]

Find an executable in the system. We will look only in the places where the operating system usually store the programs. For instance on windows we might look into "Program Files" while in linux we may look uinto "/opt or /usr/local/bin"

Parameters **program_name** – name of the program we need to look

abstract fire_admin_command_and_capture_stdout(*commands: List[Union[str, List[str]]], cwd: Optional[str] = None, env: Optional[Dict[str, Any]] = None, check_exit_code: bool = True, timeout: Optional[int] = None, log_entry: bool = False, credential_type: Optional[str] = None, credential: Optional[any] = None*) → Tuple[int, str, str]

Start a new process as admin and wait for its completition. Stdout is returned and not shown on the console

Parameters

- **commands** – command to execute
- **cwd** – directory where this command will be executed
- **env** – environment variables to set
- **check_exit_code** – if true, we will raise an exception if the command fails
- **timeout** – if after x milliseconds, the command is not yet completed

- **log_entry** – if true, we will log the command execution
- **credential_type** – type format of credentials
- **credential** – object that allows you to execute the command as an admin

abstract fire_admin_command_and_forget(*commands: List[Union[str, List[str]]]*, *cwd: Optional[str] = None*, *env: Optional[Dict[str, Any]] = None*, *log_entry: bool = False*, *credential_type: Optional[str] = None*, *credential: Optional[any] = None*) → int

Start a new process as an admin. Then do not wait for its completition. Do not show the stdout nor the stderr on screen

Parameters

- **commands** – command to execute
- **cwd** – directory where this command will be executed
- **env** – environment variables to set
- **check_exit_code** – if true, we will raise an exception if the command fails
- **timeout** – if after x milliseconds, the command is not yet completed
- **log_entry** – if true, we will log the command execution
- **credential_type** – type format of credentials
- **credential** – object that allows you to execute the command as an admin

Returns pid of the running process

abstract fire_admin_command_and_show_stdout(*commands: List[Union[str, List[str]]]*, *cwd: Optional[str] = None*, *env: Optional[Dict[str, Any]] = None*, *check_exit_code: bool = True*, *timeout: Optional[int] = None*, *log_entry: bool = False*, *credential_type: Optional[str] = None*, *credential: Optional[any] = None*) → int

Start a new process as admin and wait for its completition. Stdout is put on the console

Parameters

- **commands** – command to execute
- **cwd** – directory where this command will be executed
- **env** – environment variables to set
- **check_exit_code** – if true, we will raise an exception if the command fails
- **timeout** – if after x milliseconds, the command is not yet completed
- **log_entry** – if true, we will log the command execution
- **credential_type** – type format of credentials
- **credential** – object that allows you to execute the command as an admin

Returns error code of the program

abstract fire_admin_command_and_wait(*commands: List[Union[str, List[str]]]*, *cwd: Optional[str] = None*, *env: Optional[Dict[str, Any]] = None*, *check_exit_code: bool = True*, *timeout: Optional[int] = None*, *log_entry: bool = False*, *credential_type: Optional[str] = None*, *credential: Optional[any] = None*) → int

Start a new process and wait for its completition. Do not show the stdout nor the stderr on screen

Parameters

- **commands** – command to execute
- **cwd** – directory where this command will be executed
- **env** – environment variables to set
- **check_exit_code** – if true, we will raise an exception if the command fails
- **timeout** – if after x milliseconds, the command is not yet completed
- **log_entry** – if true, we will log the command execution
- **credential_type** – type format of credentials
- **credential** – object that allows you to execute the command as an admin

Returns error code of the program

abstract fire_command_and_capture_stdout(*commands: List[Union[str, List[str]]]*, *cwd: Optional[str] = None*, *env: Optional[Dict[str, Any]] = None*, *check_exit_code: bool = True*, *timeout: Optional[int] = None*, *log_entry: bool = False*) → Tuple[int, str, str]

Start a new process and wait for its completition. Stdout is returned and not shown on the console

Parameters

- **commands** – command to execute
- **cwd** – directory where this command will be executed
- **env** – environment variables to set
- **check_exit_code** – if true, we will raise an exception if the command fails
- **timeout** – if after x milliseconds, the command is not yet completed
- **log_entry** – if true, we will log the command execution

abstract fire_command_and_forget(*commands: List[Union[str, List[str]]]*, *cwd: Optional[str] = None*, *env: Optional[Dict[str, Any]] = None*, *log_entry: bool = False*) → int

Start a new process; then do not wait for its ocmpletition. Do not show the stdout nor the stderr on screen

Parameters

- **commands** – command to execute
- **cwd** – directory where this command will be executed
- **env** – environment variables to set
- **check_exit_code** – if true, we will raise an exception if the command fails
- **timeout** – if after x milliseconds, the command is not yet completed
- **log_entry** – if true, we will log the command execution

Returns pid of the running process

abstract fire_command_and_show_stdout(*commands: List[Union[str, List[str]]]*, *cwd: Optional[str] = None*, *env: Optional[Dict[str, Any]] = None*, *check_exit_code: bool = True*, *timeout: Optional[int] = None*, *log_entry: bool = False*) → int

Start a new process and wait for its completition. Stdout is put on the console

Parameters

- **commands** – command to execute
- **cwd** – directory where this command will be executed
- **env** – environment variables to set
- **check_exit_code** – if true, we will raise an exception if the command fails
- **timeout** – if after x milliseconds, the command is not yet completed
- **log_entry** – if true, we will log the command execution

Returns error code of the program

abstract fire_command_and_wait(*commands: List[Union[str, List[str]]]*, *cwd: Optional[str] = None*, *env: Optional[Dict[str, Any]] = None*, *check_exit_code: bool = True*, *timeout: Optional[int] = None*, *log_entry: bool = False*) → int

Start a new process and wait for its completition. Do not show the stdout nor the stderr on screen

Parameters

- **commands** – command to execute
- **cwd** – directory where this command will be executed
- **env** – environment variables to set
- **check_exit_code** – if true, we will raise an exception if the command fails
- **timeout** – if after x milliseconds, the command is not yet completed
- **log_entry** – if true, we will log the command execution

Returns error code of the program

get_current_username() → str

abstract get_env_variable(*name: str*) → str

Get an evnironment variable value. We will use the current user environment to determine the variable Raises an exception if the variable does not exist

Parameters **name** – the environment variable to fetch

Returns the environmkent variable value

abstract get_home_folder() → str

Get the absolute home folder of the current user

get_processes() → Iterable[Tuple[str, int]]

Get all the processes in execution on the system

Returns an iterable fo pairs. Each pair has 2 items: the first is the process name while the other is the pid

abstract get_program_path() → Iterable[str]

Fetch the list of paths in the PATH environment variable

is_process_with_name_running(*name: str*) → bool

Detects if there exists a process whose name contains the given string

Parameters **name** – the substring to consider

Returns True if there exists a process containing such substring, false otherwise

abstract is_program_installed(*program_name: str*) → bool
Check if a program is installed on the platform.

Parameters **program_name** – name of the program

Returns true if the program is installed on the system, false otherwise4

kill_process_with_name(*name: str*, *ignore_if_process_does_not_exists: bool = True*)
Kill the process with the given name. If the process does not exists, the function can either raises an exception or do nothing

Parameters

- **name** – the name of the process to kill
- **ignore_if_process_does_not_exists** – if true and the process does not exist, we do nothing

kill_process_with_pid(*pid: int*, *ignore_if_process_does_not_exists: bool = True*)
Kill the process with the given id. If the process does not exist

Parameters

- **pid** – the pid of the process to kill
- **ignore_if_process_does_not_exists** – if true and the process does not exist, we do nothing

ls(*folder: str*, *generate_absolute_path: bool = False*) → Iterable[str]
Show the list of all the files in the given directory

Parameters

- **folder** – folder to scan.
- **generate_absolute_path** – if true, we will generate in the outptu the absolute path of the subfolders. Otherwise we will return only the

Returns iterable of files in the directory

ls_only_directories(*folder: str*, *generate_absolute_path: bool = False*) → Iterable[str]
Show the list of all the directories in the given directory

Parameters

- **folder** – folder to scan.

- **generate_absolute_path** – if true, we will generate in the outptu the absolute path of the subfolders. Otherwise we will return only the names

Returns iterable of folders in directory

ls_only_files(*folder: str*, *generate_absolute_path: bool = False*) → Iterable[str]

Show the list of all the files (but not directories) in the given directory

Parameters

- **folder** – folder to scan.
- **generate_absolute_path** – if true, we will generate in the outptu the absolute path of the subfolders. Otherwise we will return only the

Returns iterable of files in the directory

mark_file_as_executable_by_all(*file_path: str*)

Mark the filev as executable by all

Parameters **file_path** – the file involved

mark_file_as_executable_by_owner(*file_path: str*)

Mark the filev as executable by the owner

Parameters **file_path** – the file involved

mark_file_as_readable_by_all(*file_path: str*)

Mark the file as readable by all

Parameters **file_path** – the file involved

mark_file_as_readable_by_user(*file_path: str*)

Mark the file as readable by the owner

Parameters **file_path** – the file involved

abstract set_global_environment_variable(*group_name: str*, *name: str*, *value: Any*)

Set an environment variable available for all the users on the system. This function may require a reboot in order to persistently work

Parameters

- **group_name** – name of the group the variable belongs to. May be ignored by the function implementation
- **name** – the variable name
- **value** – the variable value to set

class pmakeup.**IPMakeupCache**

Bases: abc.ABC

abstract get_name() → str
human friendly name of the cache

abstract get_variable_in_cache(*name: str*) → Any
Obtain the variable value from the cache

Parameters **name** – the name of the varaible to obtain

Returns the variable obtained

abstract has_variable_in_cache(*name: str*) → bool
Check if the variable is present in the cache

Parameters **name** – the name of the variable to check

Returns true if the variable is present inthe cache, false otherwise

abstract is_cache_present() → bool
Check if the pmakeup cache is present

abstract is_empty() → bool
Check if there is at least one variable in cache

Returns true iff there is no variables in cache

abstract reset()
Completely empty the pmakeupfile. After the operation, the cache is present, but it is empty. It is required to persistently update the cache in this method

abstract set_variable_in_cache(*name: str*, *value: Any*, *overwrites_is_exists: bool = True*)
Set a variable in the cache.

Parameters

- **name** – name of the variable to add
- **value** – value to store
- **overwrites_is_exists** – if true, we will overwrite any previous variable in the cache

abstract update_cache()
Store the cache persistently

abstract variable_names() → Iterable[str]
Set of variable names available in the cache. They are only at top level

class pmakeup.**InterestingPath**(*architecture: int*, *path: str*, *version: semantic_version.base.Version*)

Bases: abc.ABC

a path which which is important to you in some way. For example, in linux it may be the installation path of a library

exception pmakeup.**InvalidScenarioPMakeupException**
Bases: pmakeup.exceptions.PMakeupException.PMakeupException

class pmakeup.**JsonPMakeupCache**(*file_path: str*)
Bases: pmakeup.cache.IPMakeupCache.IPMakeupCache

get_name() → str
human friendly name of the cache

get_variable_in_cache(*name: str*) → Any
Obtain the variable value from the cache

Parameters **name** – the name of the varaible to obtain

Returns the variable obtained

has_variable_in_cache(*name: str*) → bool
Check if the variable is present in the cache

Parameters **name** – the name of the variable to check

Returns true if the variable is present inthe cache, false otherwise

is_cache_present() → bool
Check if the pmakeup cache is present

is_empty() → bool
Check if there is at least one variable in cache

Returns true iff there is no variables in cache

reset()
Completely empty the pmakeupfile. After the operation, the cache is present, but it is empty. It is required to persistently update the cache in this method

set_variable_in_cache(*name: str*, *value: Any*, *overwrites_is_exists: bool = True*)
Set a variable in the cache.

Parameters

- **name** – name of the variable to add
- **value** – value to store
- **overwrites_is_exists** – if true, we will overwrite any previous variable in the cache

update_cache()
Store the cache persistently

variable_names() → Iterable[str]
Set of variable names available in the cache. They are only at top level

class pmakeup.**LinuxOSSystem**(*model: pmakeup.models.PMakeupModel.PMakeupModel*)
Bases: pmakeup.platforms.IOSSystem.IOSSystem

execute_command(*commands: List[Union[str, List[str]]]*, *show_output_on_screen: bool*, *capture_stdout: bool*, *cwd: Optional[str] = None*, *env: Optional[Dict[str, Any]] = None*, *check_exit_code: bool = True*, *timeout: Optional[int] = None*, *execute_as_admin: bool = False*, *admin_password: Optional[str] = None*, *log_entry: bool = False*) → Tuple[int, str, str]

fetch_interesting_paths(*model: pmakeup.models.PMakeupModel.PMakeupModel*) → Dict[str, List[pmakeup.platforms.InterestingPath.InterestingPath]]
Fetch all the interesting paths relative to a operating system. Highly dependent on the operating system. Each path has associated different actual paths, since a single

Parameters **model** – model of the pmakeup

Returns

find_executable_in_program_directories(*program_name: str*) → Optional[str]
Find an executable in the system. We will look only in the places where the operating system usually store the programs. For instance on windows we might look into "Program Files" while in linux we may look uinto "/opt or /usr/local/bin"

Parameters **program_name** – name of the program we need to look

get_env_variable(*name: str*) → str
Get an evnironment variable value. We will use the current user environment to determine the variable Raises an exception if the variable does not exist

Parameters **name** – the environment variable to fetch

Returns the environmkent variable value

get_git_branch(*p: str*) → str

get_home_folder() → str
Get the absolute home folder of the current user

get_program_path() → Iterable[str]
Fetch the list of paths in the PATH environment variable

is_program_installed(*program_name: str*) → bool
Check if a program is installed on the platform.

Parameters **program_name** – name of the program

Returns true if the program is installed on the system, false otherwise4

is_repo_clean(*p: str*) → bool

set_global_environment_variable(*group_name: str*, *name: str*, *value: Any*)

Set an environment variable available for all the users on the system. This function may require a reboot in order to persistently work

Parameters

- **group_name** – name of the group the variable belongs to. May be ignored by the function implementation
- **name** – the variable name
- **value** – the variable value to set

class pmakeup.**LinuxPMakeupPlugin**(*model: pmakeup.models.PMakeupModel.PMakeupModel*)

Bases: pmakeup.plugins.AbstractPmakeupPlugin.AbstractPmakeupPlugin

Plugin that specifically offer methods typical of linux

test_linux(*string: str*)

Test if linux commands is loaded :param string: the string to echo'ed

class pmakeup.**LoggingPMakeupPlugin**(*model: pmakeup.models.PMakeupModel.PMakeupModel*)

Bases: pmakeup.plugins.AbstractPmakeupPlugin.AbstractPmakeupPlugin

critical(*message: str*)

Log a message using 'CRITICAL' level

Parameters **message** – the message to log

debug(*message: str*)

Log a message using 'DEBUG' level

Parameters **message** – the message to log

echo(*message: str*, *foreground: str = None*, *background: str = None*)

Print a message on the screen

Parameters

- **message** – the message to print out
- **foreground** – foreground color of the string. Accepted values: RED, GREEN, YELLOW, BLUE, MAGENT, CYAN, WHITE
- **background** – background color of the string. Accepted values: RED, GREEN, YELLOW, BLUE, MAGENT, CYAN, WHITE

echo_variables(*foreground: str = None*, *background: str = None*)
Echo all the variables defined in "variables"

Parameters

- **foreground** – the foregruodn color
- **background** – the background color

info(*message: str*)
Log a message using 'INFO' level

Parameters **message** – the message to log

print_blue(*message: str*)
Print a blue message

Parameters **message** – message to print

print_cyan(*message: str*)
Print a blue message

Parameters **message** – message to print

print_red(*message: str*)
Print a red message

Parameters **message** – message to print

print_yellow(*message: str*)
Print a blue message

Parameters **message** – message to print

class pmakeup.**OperatingSystemPMakeupPlugin**(*model: pmakeup.models.PMakeupModel.PMakeupMode*
Bases: pmakeup.plugins.AbstractPmakeupPlugin.AbstractPmakeupPlugin

current_user() → str
get the user currently logged

Returns the user currently logged

execute_admin_and_forget(*commands: Union[str, List[Union[str, List[str]]]]*, *cwd: str = None*, *env: Dict[str, Any] = None*, *check_exit_code: bool = True*, *timeout: int = None*) → int
Execute a command as admin but ensure that no stdout will be printed on the console

Parameters

- **commands** – the command to execute. They will be exeucte in the same context

- **cwd** – current working directory where the command is executed
- **env** – a dictionary representing the key-values of the environment variables
- **check_exit_code** – if true, we will generate an exception if the exit code is different than 0
- **timeout** – if positive, we will give up waiting for the command after the amount of seconds

Returns triple. The first element is the error code, the second is the stdout (if captured), the third is stderr

execute_admin_and_run_in_background(*commands: Union[str, List[Union[str, List[str]]]]*, *cwd: str = None*, *env: Dict[str, Any] = None*) → int

Execute a command as admin but ensure that no stdout will be printed on the console

Parameters

- **commands** – the command to execute. They will be exeucte in the same context
- **cwd** – current working directory where the command is executed
- **env** – a dictionary representing the key-values of the environment variables

Returns pid of running process

execute_admin_return_stdout(*commands: Union[str, List[Union[str, List[str]]]]*, *cwd: str = None*, *env: Dict[str, Any] = None*, *check_exit_code: bool = True*, *timeout: int = None*) → Tuple[int, str, str]

Execute a command as an admin. We won't show the stdout on pmakeup console but we will capture it and returned it

Parameters

- **commands** – the command to execute. They will be execute in the same context
- **cwd** – current working directory where the command is executed
- **env** – a dictionary representing the key-values of the environment variables
- **check_exit_code** – if true, we will generate an exception if the exit code is different than 0

- **timeout** – if positive, we will give up waiting for the command after the amount of seconds

Returns triple. The first element is the error code, the second is the stdout (if captured), the third is stderr

execute_admin_stdout_on_screen(*commands: Union[str, List[Union[str, List[str]]]]*, *cwd: str = None*, *env: Dict[str, Any] = None*, *check_exit_code: bool = True*, *timeout: int = None*) → int

Execute a command as an admin. We won't capture the stdout but we will show it on pmakeup console

Parameters

- **commands** – the command to execute. They will be execute in the same context
- **cwd** – current working directory where the command is executed
- **env** – a dictionary representing the key-values of the environment variables
- **check_exit_code** – if true, we will generate an exception if the exit code is different than 0
- **timeout** – if positive, we will give up waiting for the command after the amount of seconds

Returns triple. The first element is the error code, the second is the stdout (if captured), the third is stderr

execute_admin_with_password_and_run_in_background(*commands: Union[str, List[Union[str, List[str]]]]*, *password: str*, *cwd: str = None*, *env: Dict[str, Any] = None*) → int

Execute a command as admin but ensure that no stdout will be printed on the console

Parameters

- **commands** – the command to execute. They will be exeucte in the same context

- **password** – password of the user to invoke the program as an admin
- **cwd** – current working directory where the command is executed
- **env** – a dictionary representing the key-values of the environment variables

Returns triple. The first element is the error code, the second is the stdout (if captured), the third is stderr

execute_admin_with_password_fire_and_forget(*commands: Union[str, List[Union[str, List[str]]]], password: str, cwd: str = None, env: Dict[str, Any] = None, check_exit_code: bool = True, timeout: int = None*) → int

Execute a command as admin by providing the admin password. **THIS IS INCREDIBLE UNSAFE!!!!!!!!!!!!**. Please, I beg you, do **NOT** use this if you need any level of security!!!!! This will make the password visible on top, on the history, everywhere on your system. Please use it only if you need to execute a command on your local machine.

Parameters

- **commands** – the command to execute. They will be executed in the same context
- **cwd** – current working directory where the command is executed
- **env** – a dictionary representing the key-values of the environment variables
- **check_exit_code** – if true, we will generate an exception if the exit code is different than 0
- **timeout** – if positive, we will give up waiting for the command after the amount of seconds
- **password** – **[UNSAFE!!!!]** If you **really** need, you might want to run a command as an admin only on your laptop, and you want a really quick and dirty way to execute it, like as in the shell. Do **not** use this in production code, since the password will be 'printed in clear basically everywhere! (e.g., history, system monitor, probably in a file as well)

execute_admin_with_password_return_stdout(*commands: Union[str, List[Union[str, List[str]]]]*, *password: str*, *cwd: str = None*, *env: Dict[str, Any] = None*, *check_exit_code: bool = True*, *timeout: int = None*) → Tuple[int, str, str]

Execute a command as an admin. We won't show the stdout on pmakeup console but we will capture it and returned it

Parameters

- **commands** – the command to execute. They will be execute in the same context
- **password** – **[UNSAFE!!!!]** If you **really** need, you might want to run a command as an admin only on your laptop, and you want a really quick and dirty way to execute it, like as in the shell. Do **not** use this in production code, since the password will be 'printed in clear basically everywhere! (e.g., history, system monitor, probably in a file as well)
- **cwd** – current working directory where the command is executed
- **env** – a dictionary representing the key-values of the environment variables
- **check_exit_code** – if true, we will generate an exception if the exit code is different than 0
- **timeout** – if positive, we will give up waiting for the command after the amount of seconds

Returns triple. The first element is the error code, the second is the stdout (if captured), the third is stderr

execute_admin_with_password_stdout_on_screen(*commands: Union[str, List[Union[str, List[str]]]]*, *password: str*, *cwd: str = None*, *env: Dict[str, Any] = None*, *check_exit_code: bool = True*, *timeout: int = None*) → int

Execute a command as an admin. We won't capture the stdout but we will show it on pmakeup console

Parameters

- **commands** – the command to execute. They will be execute in the same context
- **password** – **[UNSAFE!!!!]** If you **really** need, you might want to run a command as an admin only on your laptop, and you want a really quick and dirty way to execute it, like as in the shell. Do **not** use this in production code, since the password will be 'printed in clear basically everywhere! (e.g., history, system monitor, probably in a file as well)
- **cwd** – current working directory where the command is executed
- **env** – a dictionary representing the key-values of the environment variables
- **check_exit_code** – if true, we will generate an exception if the exit code is different than 0
- **timeout** – if positive, we will give up waiting for the command after the amount of seconds

Returns triple. The first element is the error code, the second is the stdout (if captured), the third is stderr

execute_and_forget(*commands: Union[str, List[Union[str, List[str]]]]*, *cwd: str = None*, *env: Dict[str, str] = None*, *check_exit_code: bool = True*, *timeout: int = None*) → int

Execute a command but ensure that no stdout will be printed on the console

Parameters

- **commands** – the command to execute. They will be exeucte in the same context
- **cwd** – current working directory where the command is executed
- **env** – a dictionary representing the key-values of the environment variables
- **check_exit_code** – if true, we will generate an exception if the exit code is different than 0
- **timeout** – if positive, we will give up waiting for the command after the amount of seconds

Returns triple. The first element is the error code, the second is the stdout (if captured), the third is stderr

execute_and_run_in_background(*commands: Union[str, List[Union[str, List[str]]]]*, *cwd: str = None*, *env: Dict[str, str] = None*) → int

Execute a command but ensure that no stdout will be printed on the console

Parameters

- **commands** – the command to execute. They will be exeucte in the same context
- **cwd** – current working directory where the command is executed
- **env** – a dictionary representing the key-values of the environment variables

Returns pid of running process

execute_return_stdout(*commands: Union[str, List[Union[str, List[str]]]]*, *cwd: str = None*, *env: Dict[str, Any] = None*, *check_exit_code: bool = True*, *timeout: int = None*) → Tuple[int, str, str]

Execute a command. We won't show the stdout on pmakeup console but we will capture it and returned it

Parameters

- **commands** – the command to execute. They will be exeucte in the same context
- **cwd** – current working directory where the command is executed
- **env** – a dictionary representing the key-values of the environment variables
- **check_exit_code** – if true, we will generate an exception if the exit code is different than 0
- **timeout** – if positive, we will give up waiting for the command after the amount of seconds

Returns triple. The first element is the error code, the second is the stdout (if captured), the third is stderr

execute_stdout_on_screen(*commands: Union[str, List[Union[str, List[str]]]]*, *cwd: str = None*, *env: Dict[str, Any] = None*, *check_exit_code: bool = True*, *timeout: int = None*) → int

Execute a command. We won't capture the stdout but we will show it on pmakeup console

Parameters

- **commands** – the command to execute. They will be exeucte in the same context
- **cwd** – current working directory where the command is executed
- **env** – a dictionary representing the key-values of the environment variables
- **check_exit_code** – if true, we will generate an exception if the exit code is different than 0
- **timeout** – if positive, we will give up waiting for the command after the amount of seconds

Returns triple. The first element is the error code, the second is the stdout (if captured), the third is stderr

get_program_path() → Iterable[str]
List of paths in PATH environment variable

Returns collections of path

is_program_installed(*program_name: str*) → bool
Check if a program is reachable via commandline. We will look **only** in the PATH environment variable. If you want to look in other parts as well, conside rusing

Parameters **program_name** – the name of the program (e.g., dot)

Returns true if there is a program accessible to the PATH with the given name, false otherwise

exception pmakeup.**PMakeupException**
Bases: Exception

class pmakeup.**PMakeupModel**
Bases: abc.ABC

The application model of pmakeup progam

available_targets: Dict[str, pmakeup.TargetDescriptor.TargetDescriptor
List of available targets the given pmakeupfile provides

cli_variables: Dict[str, Any]
Variables that the user can inject from the command line. Read only

execute()
Read the Pmakefile instructions from a configured option. For example, if "input_string" is set, invoke from it. If "input_file" is set, invoke from it :return:

execute_file(*input_file: str*)
Execute the content in a file :param input_file: file containing the code to execute :return:

execute_string(*string: str*)
Execute the content of a string :param string: string to execute :return:

get_core_plugin() → *pmakeup.plugins.core.CorePMakeupPlugin.CorePMakeupPlugin*

get_files_plugin() → *pmakeup.plugins.files.FilesPMakeupPlugin.FilesPMakeupPlugin*

get_paths_plugin() → *pmakeup.plugins.paths.PathsPMakeupPlugin.PathsPMakeupPlugin*

get_plugin(*plugin: Union[str, type]*) → pmakeup.plugins.AbstractPmakeupPlugin.AbstractPmakeupP
Fetch a plugin with the given type

:param plugin. type of the plugin to look for :return: an instance of the given plugin

get_plugin_by_name(*name: str*) → pmakeup.plugins.AbstractPmakeupPlugin.AbstractPmakeupPlu
Fetch a plugin with the given type

Parameters **name** – name of the plugin to look for

Returns an instance of the given plugin

get_plugins() → Iterable[pmakeup.plugins.AbstractPmakeupPlugin.AbstractPmakeupPlugin]
get all the registered plugin up to this point

info_description: str
Description to show whenever the user wants to know what a given Pmakeupfile does

input_encoding: Optional[str]
Encoding of ::input_file

input_file: Optional[str]
File representing the "Makefile" of pmakeup

input_string: Optional[str]
String to be used in place of ::input_file

is_plugin_registered(*plugin: Union[str, pmakeup.plugins.AbstractPmakeupPlugin.AbstractPmak*
→ bool
At least one plugin instance has been initialized in the plugin graph

Parameters **plugin** – plugin to check (or plugin name)

Returns true if the plugin is already been registered in the model, false otheriwse

log_level: Optional[str]
level of the logger. INFO, DEBUG, CRITICAL

manage_pmakefile()
Main function used to programmatically call the application :return:

pmake_cache: Optional[pm.IPMakeupCache]
Cache containing data that the user wants t persist between different pmakeup runs

register_plugins(**plugin: str*)

requested_target_names: List[str]
List of targets that the user wants to perform. This list of targets are mpretty mch like the make one's (e.g., all, clean, install, uninstall)

should_show_target_help: bool
If true, we will print the information on how to use the given PMakefile

starting_cwd: str
current working directory when pamke was executed

class pmakeup.**PMakeupRegistry**
Bases: dict

The shared context that will be used when computing "eval" or "exec" function, as a global variables.

At the root there are all the functions that the developers may use.

This object holds all the data that is shared among all the pmakeup plugin

can_a_function_have_a_name(*func_name: str*) → bool

property commands
Gain access to the dictionary contianing all the registered functions

property cwd
Gain access to the CWD variable

dump_registry() → str
Fetch a JSON representaiton on the WHJOLE registry

property pmakeup_cli_variables
Gain access to the variables

property pmakeup_info
Gain access to the variables

property pmakeup_interesting_paths
Gain access to the target that the user wanted to process

property pmakeup_latest_interesting_paths
Gain access to the target that the user wanted to process

property pmakeup_model
Gain access to the variables

property pmakeup_plugins
Gain access to the set of plugins registered by the pmakeup

property pmakeup_requested_target_names
Gain access to the target that the user wanted to process

property variables
Gain access to the object containing all the variables accessible by any plugin

class pmakeup.**PathsPMakeupPlugin**(*model: pmakeup.models.PMakeupModel.PMakeupModel*)

Bases: pmakeup.plugins.AbstractPmakeupPlugin.AbstractPmakeupPlugin

abs_path(**p:* pmakeup.plugins.paths.PathsPMakeupPlugin.PathsPMakeupPlugin.path) → str

Generate a path compliant with the underlying operating system path scheme.

If the path is relative, it is relative to the cwd

Parameters **p** – the path to build

cd(**folder: str*, *create_if_not_exists: bool = True*) → str

Gain access to a directory. If the directory does nto exists, it is created If the path is relative, it is relative to the CWD

Parameters

- **folder** – folder where we need to go into
- **create_if_not_exists** – if true, we will create the directory if we try to cd into a non existent directory

Returns the directory where we have cd from

cd_into_directories(*folder: str*, *prefix: str*, *folder_format: str*, *error_if_mismatch: bool = True*)

Inside the given folder, there can be several folders, each of them with the same format. We cd into the "latest" one. How can we determine which is the "latest" one? Via folder_format. it is a string that is either: - "number": an integer number - "semver2": a semantic versionign string; We fetch the "latest" by looking at the one with the greater value. If the folder contains a folder which it is not compliant with folder_format, it is either ignored or rase error

Parameters

- **folder** – folder where several folders are located
- **prefix** – a string that prefix folder_format
- **folder_format** – either "number" or "semver2"
- **error_if_mismatch** – if a folder is not compliant with folder_format, if true we will generate an exception

Returns

change_filename_extension(*new_extension: str*, **p*) → str

Change the extension of the given path

new extensions: dat /path/to/file.txt.zp.asc -> /path/to/file.txt.zp.dat

Parameters

- **new_extension** – extension that will be set

- **p** – path to chan

Returns p, but with the updated extensions

cwd() → str

Returns the CWD the commands operates in

get_absolute_file_till_root(*filename: str*, *base: str = None*) → str

Starting from the directory base, check if a fiel called "filename" is present. If nt, recursively chekc the parent directory Raise an exception if the file is not found whern considering the root

Parameters

- **filename** – the name of the file (extension included) we need to look for
- **base** – directory where we start looking. If left missing, we consider the CWD

Returns absolute path of the file found

get_basename(**p*) → str

Compute the base name of the path

/path/to/file.txt.zp.asc -> file.txt.zp.asc

Parameters **p** – path to consider

Returns basename

get_basename_with_no_extension(**p*) → str

Compute the basename of the path and remove its extension as well

/path/to/file.txt.zp.asc -> file.txt.zp

Parameters **p** – path to consider

Returns basename

get_extension(**p*) → str

Compute the extension of a file

Parameters **p** – the file to consider

Returns the file extension

get_file_without_extension(**p: str*) → str

Compute the filename without its last extension

/path/to/some/file.txt.zip.asc –> /path/to/some/file.txt.zip

Parameters **p** – path to consider

Returns same absolute path, without extension

get_parent_directory(**p*) → str
Retrieve the absolute path of the parent directory of the specified path.

/foo/tbar/tmp.txt -> /foo/tbar

Parameters **p** – path to consider

Returns parent directory of path

get_relative_path_wrt(*p: str*, *reference: str*) → str
If we were in folder reference, what actiosn should we perform in order to reach the file p?

Parameters

- **p** – the file to reach
- **reference** – the folder we are in right now

Returns relative path

path(**p: str*) → str
Generate a path compliant wit the underlying operating system path scheme.

If the path is relative, we will **not** join it with cwd

Parameters **p** – the path to build

class pmakeup.**StringsPMakeupPlugin**(*model: pmakeup.models.PMakeupModel.PMakeupModel*)
Bases: pmakeup.plugins.AbstractPmakeupPlugin.AbstractPmakeupPlugin

match(*string: str*, *regex: str*) → bool
Check if a given string matches perfectly the given regex

Parameters

- **string** – the sting to check
- **regex** – the regex to check. The syntax is available at https://docs.python.org/3/library/re.html

Returns true if such a substring can be found, false otherwise

replace_regex_in_string(*string: str*, *regex: str*, *replacement: str*, *count: int = - 1*, *encoding: str = 'utf-8'*) → str
Replace some (or all) the occurences of a given string

If you want to use named capturing group, you can do so! For instance,

replace_regex_in_string('3435spring9437', r'(?P<word>[a-z]+)', r'aa') 'spring' will be replaced with 'springaa'

It may not work, so you can use the following syntax to achieve the same: replace_regex_in_file(file_path, '(?P<word>w+)', r'g<word>aa') 'spring' will be replaced with 'springaa'

Parameters

- **string** – string that will be involved in the replacements
- **regex** – regex to replace
- **replacement** – string that will replace *substring*
- **count** – the number of occurences to replace. -1 if you want to replace all occurences
- **encoding** – encoding used for reading the file

See https://docs.python.org/3/howto/regex.html

replace_substring_in_string(*string: str*, *substring: str*, *replacement: str*, *count: int = - 1*) → str

Replace some (or all) the occurences of a given string

Parameters

- **string** – string that will be involved in the replacements
- **substring** – the string to repplace
- **replacement** – string that will replace *substring*
- **count** – the number of occurences to replace. -1 if you want to replace all occurences

search(*string: str*, *regex: str*)

Check if a given string has a substring that matches the given regex

Parameters

- **string** – the sting to check
- **regex** – the regex to check. The syntax is available at https://docs.python.org/3/library/re.html

Returns true if such a substring can be found, false otherwise

class pmakeup.**TargetDescriptor**(*name: str*, *description: str*, *requires: Iterable[str]*, *function: Callable[], None]*)

Bases: object

class pmakeup.**TargetsPMakeupPlugin**(*model: pmakeup.models.PMakeupModel.PMakeupModel*)

Bases: pmakeup.plugins.AbstractPmakeupPlugin.AbstractPmakeupPlugin

declare_file_descriptor(*description: str*)
Defines what to write at the beginning of the info string that is displayed whenver the user wants to know what the given Pmakeupfile does

Parameters **description** – string to show

declare_target(*target_name: str*, *f: Callable[], None]*, *requires: Iterable[str] = None*, *description: str = ''*)
Declare that the user can declare a pseudo-makefile target.

Parameters

- **target_name** – name of the target to declare
- **description** – a description that is shown when listing all available targets
- **requires** – list fo target names this target requires in order to be executed. They must already exist in pmakeup environment
- **f** – the function to perform when the user requests this target

get_target_descriptor(*target_name: str*) → pmakeup.TargetDescriptor.TargetDescriptor
Get a descriptor for a given pmakeup target. Raises exception if target is not declared

Parameters **target_name** – name of the target

Returns descriptor for the target

is_target_requested(*target_name: str*) → bool
Check if the the user has specified the given target

Parameters **target_name** – the name of the target that we need to check

Returns true if the target has been declard by the user, false otherwise

process_targets()
Function used to process in the correct order. If the user requested to show the help for this file, the function will show it and return it

It will call the function declared in declare_target

class pmakeup.**TempFilesPMakeupPlugin**(*model: pmakeup.models.PMakeupModel.PMakeupModel*)
Bases: pmakeup.plugins.AbstractPmakeupPlugin.AbstractPmakeupPlugin

create_temp_directory_with(*directory_prefix: str*) → Any
Create a temporary directory on the file system where to put temporary files

Parameters **directory_prefix** – a prefix to be put before the temporary folder

Returns the absolute path of the temporary folder created. The function can be used an input of a "with" statement. The folder will be automatically removed at the end of the with.

create_temp_file(*directory: str*, *file_prefix: str = None*, *file_suffix: str = None*, *mode: str = 'r'*, *encoding: str = 'utf-8'*, *readable_for_all: bool = False*, *executable_for_owner: bool = False*, *executable_for_all: bool = False*) → str

Creates the file. You need to manually dispose of the file by yourself

Parameters

- **directory** – the directory where to put the file
- **file_prefix** – a string that will be put at the beginning of the filename
- **file_suffix** – a string that will be put at the end of the filename
- **mode** – how we will open the file. E.g., "r", "w"
- **encoding** – the encodign of the file. Default to "utf-8"
- **readable_for_all** – if True, the file can be read by anyone
- **executable_for_owner** – if True, the file can be executed by the owner
- **executable_for_all** – if True, anyone can execute the file

Returns the absolute path of the temp file

get_temp_filepath(*prefix: str = None*, *suffix: str = None*) → str

Get the filename of a temp file. You need to manually create such a temp file

Parameters

- **prefix** – a prefix the temp file to generate has
- **suffix** – a suffix the temp file to generate has

Returns the absolute path of the temp path

class pmakeup.**UtilsPMakeupPlugin**(*model: pmakeup.models.PMakeupModel.PMakeupModel*)

Bases: pmakeup.plugins.AbstractPmakeupPlugin.AbstractPmakeupPlugin

as_bool(*v: Any*) → bool

Convert a value into a boolean

Parameters **v** – value to convert as a boolean

Returns true of false

convert_table(*table_str: str*) → List[List[str]]
Convert a table printed as:

Port Type Board Name FQBN Core /dev/ttyACM1 Serial Port (USB) Arduino/Genuino MKR1000 arduino:samd:mkr1000 arduino:samd

Into a list of lists of strings

Parameters **table_str** – representation of a table

Returns list of lists of strings

get_column_of_table(*table: List[List[str]]*, *index: int*) → List[str]
Select a single column from the table, generated by ::convert_table

Parameters

- **table** – the table generated by ::convert_table
- **index** – index of the column to return. Starts from 0

Returns the column requested

get_column_of_table_by_name(*table: List[List[str]]*, *column_name: str*) → List[str]
Select a single column from the table, generated by ::convert_table We assumed the first row of the table is a header, contaiing the column names

Parameters

- **table** – the table generated by ::convert_table
- **column_name** – name of the column to return.

Returns the column requested

grep(*lines: Iterable[str]*, *regex: str*, *reverse_match: bool = False*) → Iterable[str]
Filter the lines fetched from terminal

Parameters

- **lines** – the lines to fetch
- **regex** – a python regex. If a line contains a substring which matches the given regex, the line is returned
- **reverse_match** – if True, we will return lines which do not match the pattern

Returns lines compliant with the regex

pairs(*it: Iterable[Any]*) → Iterable[Tuple[Any, Any]]
Convert the iterable into an iterable of pairs.

1,2,3,4,5,6 becomes (1,2), (2,3), (3,4), (4,5), (5,6)

Parameters **it** – iterable whose sequence we need to generate

Returns iterable of pairs

class pmakeup.**WebPMakeupPlugin**(*model: pmakeup.models.PMakeupModel.PMakeupModel*)

Bases: pmakeup.plugins.AbstractPmakeupPlugin.AbstractPmakeupPlugin

download_url(*url: str*, *destination: str = None*, *ignore_if_file_exists: bool = True*) → str

Download an artifact from internet

Parameters

- **url** – the url where the file is lcoated
- **destination** – the folder where the file will be created
- **ignore_if_file_exists** – if true, we will not perform the download at all

Returns path containing the downloaded item

class pmakeup.**WindowsOSSystem**(*model*)

Bases: pmakeup.platforms.IOSSystem.IOSSystem

fetch_interesting_paths(*model: pmakeup.models.PMakeupModel.PMakeupModel*) → Dict[str, List[pmakeup.platforms.InterestingPath.InterestingPath]]

Fetch all the interesting paths relative to a operating system. Highly dependent on the operating system. Each path has associated different actual paths, since a single

Parameters **model** – model of the pmakeup

Returns

find_executable_in_program_directories(*program_name: str*) → Optional[str]

Find an executable in the system. We will look only in the places where the operating system usually store the programs. For instance on windows we might look into "Program Files" while in linux we may look uinto "/opt or /usr/local/bin"

Parameters **program_name** – name of the program we need to look

fire_admin_command_and_capture_stdout(*commands: List[Union[str, List[str]]]*, *cwd: Optional[str] = None*, *env: Optional[Dict[str, Any]] = None*, *check_exit_code: bool = True*, *timeout: Optional[int] = None*, *log_entry: bool = False*, *credential_type: Optional[str] = None*, *credential: Optional[any] = None*) → Tuple[int, str, str]

Start a new process as admin and wait for its completition. Stdout is returned and not shown on the console

Parameters

- **commands** – command to execute
- **cwd** – directory where this command will be executed
- **env** – environment variables to set
- **check_exit_code** – if true, we will raise an exception if the command fails
- **timeout** – if after x milliseconds, the command is not yet completed
- **log_entry** – if true, we will log the command execution
- **credential_type** – type format of credentials
- **credential** – object that allows you to execute the command as an admin

fire_admin_command_and_forget(*commands: List[Union[str, List[str]]]*, *cwd: Optional[str] = None*, *env: Optional[Dict[str, Any]] = None*, *log_entry: bool = False*, *credential_type: Optional[str] = None*, *credential: Optional[any] = None*) → int

Start a new process as an admin. Then do not wait for its completition. Do not show the stdout nor the stderr on screen

Parameters

- **commands** – command to execute
- **cwd** – directory where this command will be executed
- **env** – environment variables to set

- **check_exit_code** – if true, we will raise an exception if the command fails
- **timeout** – if after x milliseconds, the command is not yet completed
- **log_entry** – if true, we will log the command execution
- **credential_type** – type format of credentials
- **credential** – object that allows you to execute the command as an admin

Returns pid of the running process

fire_admin_command_and_show_stdout(*commands: List[Union[str, List[str]]]*, *cwd: Optional[str] = None*, *env: Optional[Dict[str, Any]] = None*, *check_exit_code: bool = True*, *timeout: Optional[int] = None*, *log_entry: bool = False*, *credential_type: Optional[str] = None*, *credential: Optional[any] = None*) → int

Start a new process as admin and wait for its completition. Stdout is put on the console

Parameters

- **commands** – command to execute
- **cwd** – directory where this command will be executed
- **env** – environment variables to set
- **check_exit_code** – if true, we will raise an exception if the command fails
- **timeout** – if after x milliseconds, the command is not yet completed
- **log_entry** – if true, we will log the command execution
- **credential_type** – type format of credentials
- **credential** – object that allows you to execute the command as an admin

Returns error code of the program

fire_admin_command_and_wait(*commands: List[Union[str, List[str]]]*, *cwd: Optional[str] = None*, *env: Optional[Dict[str, Any]] = None*, *check_exit_code: bool = True*, *timeout: Optional[int] = None*, *log_entry: bool = False*, *credential_type: Optional[str] = None*, *credential: Optional[any] = None*) → int

Start a new process and wait for its completition. Do not show the stdout nor the stderr on screen

Parameters

- **commands** – command to execute
- **cwd** – directory where this command will be executed
- **env** – environment variables to set
- **check_exit_code** – if true, we will raise an exception if the command fails
- **timeout** – if after x milliseconds, the command is not yet completed
- **log_entry** – if true, we will log the command execution
- **credential_type** – type format of credentials
- **credential** – object that allows you to execute the command as an admin

Returns error code of the program

fire_command_and_capture_stdout(*commands: List[Union[str, List[str]]]*, *cwd: Optional[str] = None*, *env: Optional[Dict[str, Any]] = None*, *check_exit_code: bool = True*, *timeout: Optional[int] = None*, *log_entry: bool = False*) → Tuple[int, str, str]

Start a new process and wait for its completition. Stdout is returned and not shown on the console

Parameters

- **commands** – command to execute
- **cwd** – directory where this command will be executed
- **env** – environment variables to set
- **check_exit_code** – if true, we will raise an exception if the command fails

- **timeout** – if after x milliseconds, the command is not yet completed
- **log_entry** – if true, we will log the command execution

fire_command_and_forget(*commands: List[Union[str, List[str]]]*, *cwd: Optional[str] = None*, *env: Optional[Dict[str, Any]] = None*, *log_entry: bool = False*) → int

Start a new process; then do not wait for its ocmpletition. Do not show the stdout nor the stderr on screen

Parameters

- **commands** – command to execute
- **cwd** – directory where this command will be executed
- **env** – environment variables to set
- **check_exit_code** – if true, we will raise an exception if the command fails
- **timeout** – if after x milliseconds, the command is not yet completed
- **log_entry** – if true, we will log the command execution

Returns pid of the running process

fire_command_and_show_stdout(*commands: List[Union[str, List[str]]]*, *cwd: Optional[str] = None*, *env: Optional[Dict[str, Any]] = None*, *check_exit_code: bool = True*, *timeout: Optional[int] = None*, *log_entry: bool = False*) → int

Start a new process and wait for its completition. Stdout is put on the console

Parameters

- **commands** – command to execute
- **cwd** – directory where this command will be executed
- **env** – environment variables to set
- **check_exit_code** – if true, we will raise an exception if the command fails
- **timeout** – if after x milliseconds, the command is not yet completed
- **log_entry** – if true, we will log the command execution

Returns error code of the program

fire_command_and_wait(*commands: List[Union[str, List[str]]]*, *cwd: Optional[str] = None*, *env: Optional[Dict[str, Any]] = None*, *check_exit_code: bool = True*, *timeout: Optional[int] = None*, *log_entry: bool = False*) → int

Start a new process and wait for its completition. Do not show the stdout nor the stderr on screen

Parameters

- **commands** – command to execute
- **cwd** – directory where this command will be executed
- **env** – environment variables to set
- **check_exit_code** – if true, we will raise an exception if the command fails
- **timeout** – if after x milliseconds, the command is not yet completed
- **log_entry** – if true, we will log the command execution

Returns error code of the program

get_env_variable(*name: str*) → str

Get an evnironment variable value. We will use the current user environment to determine the variable Raises an exception if the variable does not exist

Parameters **name** – the environment variable to fetch

Returns the environmkent variable value

get_home_folder() → str

Get the absolute home folder of the current user

get_program_path() → Iterable[str]

Fetch the list of paths in the PATH environment variable

is_program_installed(*program_name: str*) → bool

Check if a program is installed on the platform.

Parameters **program_name** – name of the program

Returns true if the program is installed on the system, false otherwise4

set_global_environment_variable(*group_name: str*, *name: str*, *value: Any*)

Set an environment variable available for all the users on the system. This function may require a reboot in order to persistently work

Parameters

- **group_name** – name of the group the variable belongs to. May be ignored by the function implementation

- **name** – the variable name
- **value** – the variable value to set

class pmakeup.**WindowsPMakeupPlugin**(*model: pmakeup.models.PMakeupModel.PMakeupModel*)

Bases: pmakeup.plugins.AbstractPmakeupPlugin.AbstractPmakeupPlugin

Plugin that specifically offer methods typical of windows

add_to_regasm(*dll: str*, *architecture: int*, *regasm_exe: str = None*, *use_codebase: bool = True*, *use_tlb: bool = True*)

Add a dll into a regasm (either 32 or 64 bit) :param regasm_exe: executable of regasm. :param dll: the dll to include in the regasm :param architecture: number of bits the processor has. either 32 or 64 :param use_codebase: if set we will add /codebase :param use_tlb: if set, we will add /tlb

delete_registry(*root: str*, *key_relative_to_root: str*, *key: str*, *architecture: int = None*) → bool

Delete a key in the registry. Is not recursive

Parameters

- **root** – e.g., winreg.HKEY_CURRENT_USER
- **key_relative_to_root** – you can use "SOFTWAREMicrosoft-Internet ExplorerMain" to set it to internet explorer
- **key** – key to generate within root + key_relative_to_root
- **architecture** – architecture to user in the regedit

Returns true if the oepration suceeds, false otheriwse

delete_registry_from_current_user(*key_relative_to_root: str*, *key: str*, *architecture: int = None*) → bool

Delete a simple key-value pair in the registry. Is not recursive

Parameters

- **key_relative_to_root** – you can use "SOFTWAREMicrosoft-Internet ExplorerMain" to set it to internet explorer
- **key** – key to generate within root + key_relative_to_root
- **architecture** – architecture to user in the regedit

delete_registry_from_hkey_local_machine(*key_relative_to_root: str*, *key: str*, *architecture: int = None*) → bool

Delete a simple key-value pair in the registry. Is not recursive

Parameters

- **key_relative_to_root** – you can use "SOFTWAREMicrosoft-Internet ExplorerMain" to set it to internet explorer
- **key** – key to generate within root + key_relative_to_root
- **architecture** – architecture to user in the regedit

get_registry_current_user_values(*key: str*, *architecture: int = None*) → Iterable[Tuple[str, Any]]

get_registry_local_machine_values(*key: str*, *architecture: int = None*) → Iterable[Tuple[str, Any]]

get_registry_values(*hkey: int*, *key: str*, *architecture: int = None*) → Iterable[Tuple[str, Any, int]]

Get the values of all the key-pair items in a given key

Parameters

- **hkey** – registry root
- **key** – key to open
- **architecture** – architecture of the registry

Returns iterable of key-value pairs. The first element is the name of the keyvale, the second is the value associated. The third is the type of the value.

See https://docs.python.org/3.9/library/winreg.html?highlight=winreg#value-typeshttps://d

has_registry_current_user_value(*key: str*, *item: str*, *architecture: int = None*) → bool

Check if there exists a value in the given key

Parameters

- **key** – root key to access
- **key** – key involved
- **item** – key-vaue pair that may or may not exists
- **architecture** – architecture to use to gain access to the registry. If left missing, we will use the architecture fo the OS of the current machine

Returns true if the key-value does not exists in the given *key*

has_registry_local_machine_value(*key: str*, *item: str*, *architecture: int = None*) → bool

Check if there exists a value in the given key

Parameters

- **key** – root key to access
- **key** – key involved
- **item** – key-vaue pair that may or may not exists
- **architecture** – architecture to use to gain access to the registry. If left missing, we will use the architecture fo the OS of the current machine

Returns true if the key-value does not exists in the given *key*

has_registry_value(*hkey: int*, *key: str*, *item: str*, *architecture: int = None*) → bool

Check if there exists a value in the given key

Parameters

- **hkey** – root key to access
- **key** – key involved
- **item** – key-vaue pair that may or may not exists

Returns true if the key-value does not exists in the given *key*

publish_dotnet_project(*cwd: str*, *runtime: str*, *configuration: str*, *solution_directory: str*) → None

publish a dotnet project. For example:

echo start "PUBLISHING RUNEXTERNALLY" /D "$(SolutionDir)xplan-subsystem-topsh
dotnet publish –runtime "$(PublishRuntime)" –configuration "$(PublishConfiguration)" /p:SolutionDir="$(SolutionDir)"

Parameters

- **cwd** – directory where to call the dotnet publish
- **runtime** – runtime that you will use to publish. Allowed values are 'x86' or 'x64'
- **configuration** – configuration used to build the artifact. Allowed values are 'Debug' or 'Release'
- **solution_directory** – directory containing the a .sln file containing the project that you need to build

read_registry_current_user_value(*key: str*, *item: str*, *architecture: int = None*) → Any

read_registry_local_machine_value(*key: str*, *item: str*, *architecture: int = None*) → Any

read_registry_value(*hkey: int*, *key: str*, *item: str*, *architecture: int = None*) → Any

Get the value associated to a single key-pair value in the given key

Parameters

- **key** – key to open
- **item** – name of the key-value pair to obtain
- **architecture** – architecture of the registry to connect to

Returns value associated to the item

remove_from_regasm(*dll: str*, *architecture: int*, *regasm_exe: str = None*, *use_codebase: bool = True*, *use_tlb: bool = True*)

Remove a dll into a regasm (either 32 or 64 bit) :param regasm_exe: executable of regasm. :param dll: the dll to include in the regasm :param architecture: number of bits the processor has. either 32 or 64 :param use_codebase: if set we will add /codebase :param use_tlb: if set, we will add /tlb

set_registry(*root: str*, *key_relative_to_root: str*, *key: str*, *value_type*, *value: Any*, *architecture: int = None*) → bool

Set a key in the registry :param root: e.g., winreg.HKEY_CURRENT_USER :param key_relative_to_root: you can use "SOFTWAREMicrosoftInternet ExplorerMain" to set it to internet explorer :param key: key to generate within root + key_relative_to_root :param value_type: type of the vlaue to create :param value: value to set :param architecture: architecture to user in the regedit :return: true if the oepration suceeds, false otheriwse

set_registry_as_int(*root: str*, *key_relative_to_root: str*, *key: str*, *value: int*, *architecture: int = None*) → bool

Set a key which is an int.

Parameters

- **root** – e.g., winreg.HKEY_CURRENT_USER
- **key_relative_to_root** – you can use "SOFTWAREMicrosoft-Internet ExplorerMain" to set it to internet explorer
- **key** – key to generate within root + key_relative_to_root
- **value** – value to set
- **architecture** – architecture to user in the regedit

set_registry_as_string(*root: str*, *key_relative_to_root: str*, *key: str*, *value: str*, *architecture: int = None*) → bool

Set a key which is an int.

Parameters

- **root** – e.g., winreg.HKEY_CURRENT_USER

- **key_relative_to_root** – you can use "SOFTWAREMicrosoft-Internet ExplorerMain" to set it to internet explorer
- **key** – key to generate within root + key_relative_to_root
- **value** – value to set
- **architecture** – architecture to user in the regedit

set_registry_in_current_user_as_int(*key_relative_to_root: str*, *key: str*, *value: int*, *architecture: int = None*) → bool

Set a key which is an int inside a current user

Parameters

- **key_relative_to_root** – you can use "SOFTWAREMicrosoft-Internet ExplorerMain" to set it to internet explorer
- **key** – key to generate within root + key_relative_to_root
- **value** – value to set
- **architecture** – architecture to user in the regedit

set_registry_in_current_user_as_string(*key_relative_to_root: str*, *key: str*, *value: str*, *architecture: int = None*) → bool

Set a key which is an int inside a current user

Parameters

- **key_relative_to_root** – you can use "SOFTWAREMicrosoft-Internet ExplorerMain" to set it to internet explorer
- **key** – key to generate within root + key_relative_to_root
- **value** – value to set
- **architecture** – architecture to user in the regedit

set_registry_in_local_machine_as_string(*key_relative_to_root: str*, *key: str*, *value: str*, *architecture: int = None*) → bool

Set a key which is an int inside a current user

Parameters

- **key_relative_to_root** – you can use "SOFTWAREMicrosoft-Internet ExplorerMain" to set it to internet explorer
- **key** – key to generate within root + key_relative_to_root
- **value** – value to set

- **architecture** – architecture to user in the regedit

test_windows(*string: str*)
Test if windows commands is loaded :param string: the string to echo'ed

pmakeup.**path**
alias of str

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

A

C

D

E

F

G

H

I

J

K

L

M

Q

R

S

T

U

V

W