update-dotdee

Release 6.0

May 17, 2020

Contents

Welcome to the documentation of *update-dotdee* version 6.0! The following sections are available:

CHAPTER 1

User documentation

The readme is the best place to start reading, it's targeted at all users and documents the command line interface:

1.1 update-dotdee: Generic modular configuration file manager

The update-dotdee program makes it easy to manage configuration files with modular contents in the style of Debian and dotdee. The program takes the pathname of a configuration file and updates that file based on the contents of the files in the directory with the same name as the file but ending in `.d`. It's currently tested on cPython 2.7, 3.5+ and PyPy (2.7).

1.1.1 Installation

The *update-dotdee* package is available on PyPI which means installation should be as simple as:

```
$ pip install update-dotdee
```

There's actually a multitude of ways to install Python packages (e.g. the per user site-packages directory, virtual environments or just installing system wide) and I have no intention of getting into that discussion here, so if this intimidates you then read up on your options before returning to these instructions ;-).

1.1.2 Usage

There are two ways to use the update-dotdee package: As the command line program `update-dotdee` and as a Python API. For details about the Python API please refer to the API documentation available on Read the Docs. The command line interface is described below.

Usage: *update-dotdee FILENAME*

Generate a (configuration) file based on the contents of the files in the directory with the same name as FILENAME but ending in '.d'.

If FILENAME exists but the corresponding directory does not exist yet, the directory is created and FILENAME is moved into the directory so that its existing contents are preserved.

Supported options:

Option	Description
`-f`, `--force`	Update FILENAME even if it contains local modifications, instead of aborting with an error message.
`-u`, `--use-sudo`	Enable the use of "sudo" to update configuration files that are not readable and/or writable for the current user (or the user logged in to a remote system over SSH).
`-r`, `--remote-host=SSH_ALIAS`	Operate on a remote system instead of the local system. The `SSH_ALIAS` argument gives the SSH alias of the remote host.
`-v`, `--verbose`	Increase logging verbosity (can be repeated).
`-q`, `--quiet`	Decrease logging verbosity (can be repeated).
`-h`, `--help`	Show this message and exit.

1.1.3 Example

The /etc/hosts file is a simple example of a configuration file that can be managed using update-dotdee. Individual files in the `/etc/hosts.d` directory contain snippets that are added to the configuration file on each run. For example:

```
peter@macbook> sudo update-dotdee /etc/hosts
2013-07-06 19:32:03 macbook INFO Reading file: /etc/hosts.d/1-local
2013-07-06 19:32:03 macbook INFO Reading file: /etc/hosts.d/2-work
2013-07-06 19:32:03 macbook INFO Reading file: /etc/hosts.d/3-ipv6
2013-07-06 20:59:24 macbook INFO Checking for local changes to /etc/hosts
2013-07-06 19:32:03 macbook INFO Writing file: /etc/hosts
```

1.1.4 How it works

Some notes about how update-dotdee works:

- If the given file exists but the corresponding directory does not exist yet, the directory is created and the file is moved into the directory (and renamed to `local`) so that its existing contents are preserved.

- If the generated file has been modified since the last run, update-dotdee will refuse to overwrite its contents (unless you use the `-f` or `--force` option).
- The files in the `.d` directory are concatenated in the natural sorting order of the filenames (as implemented by the naturalsort package).
- Executable files in the `.d` directory are executed and their standard output is incorporated into the generated contents (since version 4.0).

1.1.5 Use cases

Here are some example use cases for update-dotdee:

SSH client configuration The update-dotdee program was created in 2013 to provide modular SSH client configurations. It was used to generate the `~/.ssh/config` file from the contents of the files in the `~/.ssh/config.d` directory. This functionality was needed because I developed an SSH client configuration generator based on a database of server metadata and I was looking for a way to update the user's `~/.ssh/config` without trashing the existing (carefully handcrafted) contents.

System wide configuration files Linux system configuration files like /etc/crypttab, /etc/fstab and /etc/hosts lack modularity and manipulating them using command line tools like awk and sed can be fragile and/or become unwieldy :-). However if you can get your configuration sources (for example Ansible playbooks, Debian packages and manual configuration) to agree on the use of update-dotdee then you have an elegant, robust and predictable alternative.

1.1.6 Read only alternative

Sometimes the use of update-dotdee or a similar mechanism is the only way to get multiple configuration sources to cooperate, but it is a bit of a heavyweight solution. For the Python packages that I've published I wanted a more lightweight alternative that simply searches for and loads `*.ini` configuration files. This is why ConfigLoader was added in release 5.0.

1.1.7 Contact

The latest version of update-dotdee is available on PyPI and GitHub. The documentation is hosted on Read the Docs and includes a changelog. For bug reports please create an issue on GitHub. If you have questions, suggestions, etc. feel free to send me an e-mail at peter@peterodding.com.

1.1.8 License

This software is licensed under the MIT license.

© 2020 Peter Odding.

CHAPTER 2

API documentation

The following API documentation is automatically generated from the source code:

2.1 API documentation

The following API documentation was automatically generated from the source code of *update-dotdee* 6.0:

2.1.1 `update_dotdee`

Generic modular configuration file management.

The `update_dotdee` module provides two classes that implement alternative strategies for using modular configuration files:

- `UpdateDotDee` implements the Python API of the `update-dotdee` program which can be used to split a monolithic configuration file into a directory of files with configuration snippets. The monolithic configuration file is updated by concatenating the files with configuration snippets to enable support for programs that only handle a single configuration file.
- `ConfigLoader` is a lightweight alternative for `UpdateDotDee` that makes it easy for Python programs to load `*.ini` configuration files from multiple locations including `.d` directories. It doesn't generate any files, it just finds and loads them.

class `update_dotdee.`**`UpdateDotDee`**(***kw*)

The `UpdateDotDee` class implements the Python API of *update-dotdee*.

To create an *UpdateDotDee* object you need to provide a value for the *filename* property. You can set the values of properties by passing keywords arguments to the initializer, for details refer to the documentation of the PropertyManager superclass.

Here's an overview of the *UpdateDotDee* class:

Superclass:	PropertyManager
Public methods:	*execute_file()*, *read_file()*, *update_file()* and *write_file()*
Properties:	*checksum_file*, *context*, *directory*, *filename*, *force*, *new_checksum* and *old_checksum*

When you initialize a *UpdateDotDee* object you are required to provide a value for the *filename* property. You can set the values of the *checksum_file*, *context*, *directory*, *filename* and *force* properties by passing keyword arguments to the class initializer.

checksum_file
The pathname of the file that stores the checksum of the generated file (a string).

Note: The *checksum_file* property is a mutable_property. You can change the value of this property using normal attribute assignment syntax. To reset it to its default (computed) value you can use del or delattr().

context
An execution context created by executor.contexts.

Defaults to a LocalContext object.

Note: The *context* property is a custom_property. You can change the value of this property using normal attribute assignment syntax. This property's value is computed once (the first time it is accessed) and the result is cached. To clear the cached value you can use del or delattr().

directory
The pathname of the directory with configuration snippets (a string).

Note: The *directory* property is a mutable_property. You can change the value of this property using normal attribute assignment syntax. To reset it to its default (computed) value you can use del or delattr().

filename
The pathname of the configuration file to generate (a string).

Note: The *filename* property is a required_property. You are required to provide a value for this property by calling the constructor of the class that defines the property with a keyword argument named *filename* (unless a custom constructor is defined, in this case please refer to the documentation of that constructor). You can change the value of this property using normal attribute assignment syntax.

force
True to overwrite modified files, False to abort (the default).

Note: The *force* property is a `mutable_property`. You can change the value of this property using normal attribute assignment syntax. To reset it to its default (computed) value you can use `del` or `delattr()`.

new_checksum
Get the SHA1 digest of the contents of *filename* (a string).

old_checksum
Get the checksum stored in *checksum_file* (a string or `None`).

update_file (*force=None*)
Update the file with the contents of the files in the `.d` directory.

Parameters **force** – Override the value of *force* (a boolean or `None`).

Raises *RefuseToOverwrite* when *force* is `False` and the contents of *filename* were modified.

read_file (*filename*)
Read a text file and provide feedback to the user.

Parameters **filename** – The pathname of the file to read (a string).

Returns The contents of the file (a string).

execute_file (*filename*)
Execute a file and provide feedback to the user.

Parameters **filename** – The pathname of the file to execute (a string).

Returns Whatever the executed file returns on stdout (a string).

write_file (*filename*, *contents*)
Write a text file and provide feedback to the user.

Parameters

- **filename** – The pathname of the file to write (a string).
- **contents** – The new contents of the file (a string).

class update_dotdee.**ConfigLoader** (***kw*)
Wrapper for `configparser` that searches `*.d` directories.

The *ConfigLoader* class is a simple wrapper for `configparser` that searches for `*.ini` configuration files in system-wide and/or user-specific configuration directories:

- In normal usage the caller is expected to set *program_name* and let *ConfigLoader* take care of details like searching for available configuration files.
- Alternatively the caller can set *available_files* to bypass the usage of *program_name*, *base_directories* and *filename_extension* to generate *filename_patterns*.

The *parser* and *section_names* properties and the *get_options()* method provide access to the configuration.

Here's an overview of the *ConfigLoader* class:

Superclass:	`PropertyManager`
Public methods:	*`get_main_pattern()`*, *`get_modular_pattern()`*, *`get_options()`*, *`get_prefix()`* and *`report_issue()`*
Properties:	*`available_files`*, *`base_directories`*, *`documentation`*, *`filename_extension`*, *`filename_patterns`*, *`parser`*, *`program_name`*, *`section_names`* and *`strict`*

You can set the values of the *`available_files`*, *`base_directories`*, *`filename_extension`*, *`filename_patterns`*, *`program_name`* and *`strict`* properties by passing keyword arguments to the class initializer.

`available_files`

The filenames of the available configuration files (a list of strings).

The value of *`available_files`* is computed the first time its needed by searching for available configuration files that match *`filename_patterns`* using `glob()`. If you set *`available_files`* this effectively disables searching for configuration files.

Note: The *`available_files`* property is a `custom_property`. You can change the value of this property using normal attribute assignment syntax. This property's value is computed once (the first time it is accessed) and the result is cached. To clear the cached value you can use `del` or `delattr()`.

`base_directories`

The directories that are searched for configuration files (a list of strings).

By default this list contains three entries in the following order:

Directory	Description
`/etc`	The directory for system wide configuration files on Unix like operating systems.
`~`	The profile directory of the current user (also available as the environment variable `$HOME`).
`~/.config`	Alternative directory for user specific configuration files (also known as $XDG_CONFIG_HOME).

The order of these entries is significant because it defines the order in which configuration files are loaded by *`parser`* which controls how overrides work (when multiple files are loaded).

In this order, user specific configuration files override system wide configuration files. The reasoning behind this is that the operator may not be in a position to change system wide configuration files, even though this is an important use case to support.

Note: The *`base_directories`* property is a `custom_property`. You can change the value of this property using normal attribute assignment syntax. This property's value is computed once (the first time it is accessed) and the result is cached. To clear the cached value you can use `del` or `delattr()`.

`documentation`

Configuration documentation in reStructuredText syntax (a string).

The purpose of the *`documentation`* property is to provide documentation on the integration of *`ConfigLoader`* into other projects without denormalizing the required knowledge via copy/paste.

Note: The *documentation* property is a `cached_property`. This property's value is computed once (the first time it is accessed) and the result is cached. To clear the cached value you can use `del` or `delattr()`.

filename_extension

The filename extension of configuration files (a string, defaults to `.ini`).

Note: The *filename_extension* property is a `mutable_property`. You can change the value of this property using normal attribute assignment syntax. To reset it to its default (computed) value you can use `del` or `delattr()`.

filename_patterns

Filename patterns to search for available configuration files (a list of strings).

The value of *filename_patterns* is computed the first time it is needed. Each of the *base_directories* generates two patterns:

1. A pattern generated by *get_main_pattern()*.
2. A pattern generated by *get_modular_pattern()*.

Here's an example:

```
>>> from update_dotdee import ConfigLoader
>>> loader = ConfigLoader(program_name='update-dotdee')
>>> loader.filename_patterns
['/etc/update-dotdee.ini',
 '/etc/update-dotdee.d/*.ini',
 '~/.update-dotdee.ini',
 '~/.update-dotdee.d/*.ini',
 '~/.config/update-dotdee.ini',
 '~/.config/update-dotdee.d/*.ini']
```

Note: The *filename_patterns* property is a `custom_property`. You can change the value of this property using normal attribute assignment syntax. This property's value is computed once (the first time it is accessed) and the result is cached. To clear the cached value you can use `del` or `delattr()`.

parser

A `configparser.RawConfigParser` object with *available_files* loaded.

Note: The *parser* property is a `custom_property`. This property's value is computed once (the first time it is accessed) and the result is cached. To clear the cached value you can use `del` or `delattr()`.

program_name

The name of the application whose configuration we're managing (a string).

The value of this property is used by *filename_patterns* to generate filenames of configuration files and directories.

Note: The *program_name* property is a `mutable_property`. You can change the value of this property using normal attribute assignment syntax. To reset it to its default (computed) value you can use del or delattr().

section_names

The names of the available sections (a list of strings).

Note: The *section_names* property is a cached_property. This property's value is computed once (the first time it is accessed) and the result is cached. To clear the cached value you can use del or delattr().

strict

Whether to be strict or forgiving when something goes wrong (a boolean).

When *strict* is True and something goes wrong an exception will be raised, whereas if it is False (the default) a warning message will be logged but no exception is raised.

Note: The *strict* property is a mutable_property. You can change the value of this property using normal attribute assignment syntax. To reset it to its default (computed) value you can use del or delattr().

get_main_pattern(*directory*)

Get the glob() pattern to find the main configuration file.

Parameters **directory** – The pathname of a base directory (a string).

Returns A filename pattern (a string).

This method generates a pattern that matches a filename based on *program_name* with the suffix *filename_extension* in the given base *directory*. Here's an example:

```
>>> from update_dotdee import ConfigLoader
>>> loader = ConfigLoader(program_name='update-dotdee')
>>> [loader.get_main_pattern(d) for d in loader.base_directories]
['/etc/update-dotdee.ini',
 '~/.update-dotdee.ini',
 '~/.config/update-dotdee.ini']
```

get_modular_pattern(*directory*)

Get the glob() pattern to find modular configuration files.

Parameters **directory** – The pathname of a base directory (a string).

Returns A filename pattern (a string).

This method generates a pattern that matches a directory whose name is based on *program_name* with the suffix .d containing files matching the configured *filename_extension*. Here's an example:

```
>>> from update_dotdee import ConfigLoader
>>> loader = ConfigLoader(program_name='update-dotdee')
>>> [loader.get_modular_pattern(d) for d in loader.base_directories]
['/etc/update-dotdee.d/*.ini',
 '~/.update-dotdee.d/*.ini',
 '~/.config/update-dotdee.d/*.ini']
```

get_options(*section_name*)

Get the options defined in a specific section.

Parameters **section_name** – The name of the section (a string).

Returns A `dict` with options.

`get_prefix`(*directory*)
Get the filename prefix for the given base directory.

Parameters **`directory`** – The pathname of a directory (a string).

Returns The string '.' for the user's profile directory, an empty string otherwise.

`report_issue`(*message*, **args*, ***kw*)
Handle a problem by raising an exception or logging a warning (depending on *`strict`*).

exception `update_dotdee.`**`RefuseToOverwrite`**
Raised when *update-dotdee* notices that a generated file was modified.

`update_dotdee.`**`inject_documentation`**(***options*)
Generate configuration documentation in reStructuredText syntax.

Parameters **`options`** – Any keyword arguments are passed on to the *`ConfigLoader`* initializer.

This methods injects the generated documentation into the output generated by cog.

2.1.2 `update_dotdee.cli`

Usage: *update-dotdee FILENAME*

Generate a (configuration) file based on the contents of the files in the directory with the same name as FILENAME but ending in '.d'.

If FILENAME exists but the corresponding directory does not exist yet, the directory is created and FILENAME is moved into the directory so that its existing contents are preserved.

Supported options:

Option	Description
`-f`, `--force`	Update FILENAME even if it contains local modifications, instead of aborting with an error message.
`-u`, `--use-sudo`	Enable the use of "sudo" to update configuration files that are not readable and/or writable for the current user (or the user logged in to a remote system over SSH).
`-r`, `--remote-host=SSH_ALIAS`	Operate on a remote system instead of the local system. The `SSH_ALIAS` argument gives the SSH alias of the remote host.
`-v`, `--verbose`	Increase logging verbosity (can be repeated).
`-q`, `--quiet`	Decrease logging verbosity (can be repeated).
`-h`, `--help`	Show this message and exit.

`update_dotdee.cli.`**`main`**()
Command line interface for the `update-dotdee` program.

CHAPTER 3

Change log

The change log lists notable changes to the project:

3.1 Changelog

The purpose of this document is to list all of the notable changes to this project. The format was inspired by Keep a Changelog. This project adheres to semantic versioning.

3.1.1 Release 6.0 (2020-05-17)

This is a "maintenance release" that updates Python compatibility:

- Python 3.7 and 3.8 are now officially supported.
- Python 2.6 and 3.4 are no longer supported.

Lots of miscellaneous changes sneaked in, no real code changes though:

- Changed the readme to use console highlighting.
- Documentation improvements, added this changelog.
- Fixed humanfriendly 8 deprecation warnings.
- Changed makefile to use Python 3 for local development.
- Added `license=MIT` and `python_requires` to setup script.

3.1.2 Release 5.0 (2018-03-29)

Add support for `*.ini` configuration file loading, for details refer to the new `ConfigLoader` class.

3.1.3 Release 4.0 (2018-03-25)

Merged pull request #2 adding the ability to execute files and use their output.

3.1.4 Release 3.0 (2017-07-13)

Switch to `executor.contexts` to allow for remote execution.

3.1.5 Release 2.0 (2017-07-13)

Cleaner Python API, separate command line interface.

Detailed changes:

- Refactor setup script, add wheel support.
- Refactor all the things! (Coveralls, Read the Docs, Travis CI, tox, pytest, a test suite, code style checks, ...).
- Separate command line interface from Python API.

3.1.6 Release 1.1 (2017-07-12)

Merged pull request #1 adding support for Python 3.

3.1.7 Release 1.0.10 (2014-05-06)

Make `RefuseToOverwrite` error message user friendly (explain how to proceed).

3.1.8 Release 1.0.9 (2013-09-08)

Made the version pinning of requirements less strict.

3.1.9 Release 1.0.8 (2013-08-06)

Started using `coloredlogs.install()`.

3.1.10 Release 1.0.7 (2013-08-06)

Bumped coloredlogs requirement to 0.4.3.

3.1.11 Release 1.0.6 (2013-07-21)

- Added (absolute) version pinning to requirements.
- Moved version number from `setup.py` to `update_dotdee`.

3.1.12 Release 1.0.5 (2013-07-16)

Extracted directory creation out into a separate method.

3.1.13 Release 1.0.4 (2013-07-15)

Moved log handler initialization to `main()`.

3.1.14 Release 1.0.3 (2013-07-08)

Improved the documentation (e.g. documented natural order sorting).

3.1.15 Release 1.0.2 (2013-07-08)

Bug fix: Ignore checksum on the first (migration) run which moves the target file into the source directory.

3.1.16 Release 1.0.1 (2013-07-07)

Moved logging initialization out of "user accessible" code which can be run multiple times and should not cause log duplication.

3.1.17 Release 1.0 (2013-07-06)

The first release didn't amount to more than a hundred lines of Python code, but it did what it was supposed to do (generate a single text file by concatenating a directory of text files together).

Python Module Index

Index