Watch Do Documentation

Release 1.1.2

Vimist

Jun 02, 2020

Contents:

Watch Do is primarily a command line utility that allows you to monitor files for changes using a variety of different methods (`MD5` hash of the file, `ModificationTime`, etc) and then perform actions based on these changes.

The core Watch Do libraries can be used externally from the command line utility to provide similar functionality for use in other scripts and programs.

CHAPTER 1

Installation

To install Watch Do, ensure you have pip installed using your distributions package manager and then run the following command:

```
pip install git+https://github.com/vimist/watch-do
```

CHAPTER 2

Basic Usage

You can start making use of Watch Do right away! A basic Watch Do command can be seen below, this watches all `.py` files recursively using the default watcher (*`ModificationTime`*) and then runs `make test` in the directory that Watch Do was launched in.

```
watch-do -w '**/*.py' -d 'make test'
```

Run `watch-do --help` for more information on what all of the different command line switches do.

Note: The `-r` (`--reglob`) switch is often useful to maintain an up-to-date list of files that trigger the doers to run.

CHAPTER 3

Modules

This is the list of modules that Watch Do makes use of. All modules are documented to make it easy to integrate within your own project.

3.1 Modules

This is the list of modules that Watch Do makes use of. All modules are documented to make it easy to integrate within your own project.

3.1.1 Watchers

Watchers implement methods of determining if a file has changed.

All watchers inherit from the base *Watcher* class. This allows the derived class to focus on performing the change detection rather than having to implement core functionality.

The Base Class

The *Watcher* base class is responsible for providing the high level interface to a watcher, the actual functionality is left to the derived class.

The watchers are typically created and managed by an instance of a *WatcherManager* class.

> **Warning:** This class cannot be instantiated directly, it is an abstract base class. Only derived classes that inherit from this class and implement `_get_value()` can be instantiated.

class `watch_do.watchers.watcher.`**`Watcher`**(*file_name*)

This is the base *Watcher* that all other watchers should inherit from.

A file name is passed in that will be monitored for changes.

Note: The file state is only checked when the *has_changed()* method is called.

Initialise the *Watcher*.

Parameters **file_name** (*str*) – The file path that the watcher should detect changes for.

_get_value()
Get the current value of the watched file.

Attention: This method should be overwritten and implemented in child classes.

This method determines the current change value of the file being watched. This could be the file's hash, the modified time, or some other value that can be used to determine if we should report the file as changed on the next call to *has_changed()*.

Returns A value representing the current state of the object that this base class can use to determine if the file has changed.

Return type str

file_name
Get the name and path of the file that this watcher is monitoring.

has_changed()
Determine if the file has changed since the last call to this method.

Warning: The first call to this method will **always** return False.

Returns A boolean, indicating if the watched file has changed.

Return type bool

Built-In Watchers

These are the built-in watchers that were available for use at the time this documentation was built.

Note: The watchers below all inherit from the above *Watcher* class. This means that all methods and properites detailed above are also available on these classes below even though they aren't mentioned.

class watch_do.watchers.**MD5**(*file_name*)
MD5 hash based change detection.

This class uses MD5 hashes based on the files contents to enable change detection.

Initialise the *Watcher*.

Parameters **file_name** (*str*) – The file path that the watcher should detect changes for.

class watch_do.watchers.**ModificationTime**(*file_name*)
A modification time based watcher.

This class uses the files modification time to enable change detection.

Initialise the *Watcher*.

Parameters **file_name** (*str*) – The file path that the watcher should detect changes for.

3.1.2 Doers

Doers implement interfaces that enable the performing of actions.

All doers inherit from the base *Doer* class. This allows the derived class to focus on actually executing the action rather than having to implement core functionality.

The Base Class

The *Doer* base class is responsible for providing the high level interface to a doer, the actual functionality is left to the derived class.

The doers are typically created and managed by an instance of a *DoerManager* class.

Warning: This class cannot be instantiated directly, it is an abstract base class. Only derived classes that inherit from this class and implement `run()` can be instantiated.

class watch_do.doers.doer.**Doer**(*command*)

This is the base *Doer* that all other doers should inherit from.

A command is passed in that will determine the action that should be performed.

Initialise the *Doer*.

Parameters **command** (*str*) – The command that details what action should be performed.

static _interpolate_file_name(*string*, *file_name*)

Interpolate the `file_name` into a given `string`.

The `string` parameter will be searched for `%f` and replaced with `file_name`. Any escaped `%f`'s will be unescaped and ignored (i.e. `\%f` becomes `%f`).

Parameters

- **string** (*str*) – The string to interpolate the `file_name` into.
- **file_name** (*str*) – The file name to insert into the `string`.

Returns The input string with file name interpolated.

Return type str

command

Get the command this doer is performing.

run(*file_name*)

Run the doer against a specific file.

This method runs the command passed into the constructor against a specific file.

Parameters **file_name** (*str*) – The file name to run this doer against.

Yields *str* –

A string containing the output (possibly the partial output) of the command, both stdout and stderr.

Built-In Doers

These are the built-in doers that were available for use at the time this documentation was built.

Note: The doers below all inherit from the above *Doer* class. This means that all methods and properites detailed above are also available on these classes below even though they aren't mentioned.

The *Shell* class provides a method to run shell commands and capture their output.

As an example, the following code would provide a method of getting the output from listing a specific files attributes on the command line.

```
>>> doer = Shell('ls -lh "%f"')
```

To actually run and retrieve the output of this command, the `run()` method should be called.

```
>>> doer.run('myfile.txt')
```

class watch_do.doers.shell.**Shell**(*command*)

Interface with a shell to allow running standard shell commands.

This doer enables commands to be run in a shell and have the output captured.

Initialise the *Doer*.

Parameters **command** (*str*) – The command that details what action should be performed.

run(*file_name*)

Run the command in the shell.

The `_interpolate_file_name()` is called on the `command` with `file_name` as a parameter to ensure this `file_name` is interpolated if it's required.

Parameters **file_name** (*str*) – The `file_name` that this doer should run against.

Yields *str* –

A string containing the output (possibly the partial output) of the command, both stdout and stderr.

3.1.3 Doer Manager

The *DoerManager* class is responsible for orchestrating the doers.

Commands for the different types of doers, for example the *Shell* doer, are provided to this class, which are then parsed and converted to their respective doer instances. The default doer is also taken into account for commands that don't explicitly specify a doer.

As an example, the following code would parse out the command specified as the first argument and create a *Shell* doer from it.

```
>>> manager = DoerManager(['shell:echo "%f changed!"'], Shell)
```

In the above case, the `shell:` prefix wasn't necessary, as the default doer (the second argument) was already set to *Shell*.

All of the doers can be run by calling the *run_doers()* method.

```
>>> manager.run_doers('my_file.txt')
```

class watch_do.doer_manager.**DoerManager**(*commands*, *default_doer*)

This class creates and manages doers.

Commands are passed in, which then get parsed and converted to instances of doers. All doers can be run using the *run_doers()* method with relevant output returned.

Initialise the *DoerManager* and parse all commands.

The commands that get passed into this class are parsed (removing their doer: prefix if required) and have the relevant doer instances created for them.

Parameters

- **commands** (*list*) – A list of strings containing the commands to create doers for. Each command (str) in the list of commands should be prefixed with doer:, where 'doer' is the name of the doer (i.e. shell). If the command is not prefixed with doer: the default_doer will be used.
- **default_doer** (*Doer*) – A reference to a doer class to use as the default doer if one is not explicitly specified using the doer: prefix.

commands

list – The list of stings that this *DoerManager* is managing.

default_doer

Doer – The doer that is used if one is explicitly specified in the command.

doers

list – The doers that were created as a result of passing the commands.

run_doers(*file_name*)

Run each doer in turn and yield its output.

Yields *str* –

A string that contains the combined output of stdout and stderr from the doers.

3.1.4 Watcher Manager

The *WatcherManager* class is responsible for orchestrating the watchers.

The watchers are created based on the files returned by the *GlobManager* instance that get provided to this class. For each file that the *GlobManager* returns a new *Watcher* is created. Multiple options can be provided to this class that allows for some configuration, please see the __init__() method documentation for details.

As an example, the following code would set up watchers for all files returned by glob_manager using the *ModificationTime* method. Newly created files are detected and added to the watch list and files that get deleted are considered to be a change (this is specified as the last two parameters of the constructor).

```
>>> manager = WatcherManager(
...     ModificationTime, glob_manager, True, True)
```

All of the changed files can be retrieved by calling get_changed_files().

```
>>> manager.get_changed_files()
```

class watch_do.watcher_manager.**WatcherManager**(*watcher*, *glob_manager*, *reglob*, *changed_on_remove*)

This class creates and manages watchers.

A *Watcher* and an instance of *GlobManager* are passed in, which provides the necessary information for the class to create the required watchers that can be used to detect changes.

Initialise the *WatcherManager*.

Parameters

- **watcher** (*Watcher*) – A reference to a subclass of *Watcher* (i.e. *ModificationTime*) that will be used watch the files provided by the *GlobManager*.
- **glob_manager** (*GlobManager*) – The glob manager responsible for providing a list of files.
- **reglob** (*bool*) – A boolean value indicating whether to re-evaluate globs when *get_changed_files()* is called.
- **changed_on_remove** (*bool*) – A boolean value indicating whether to consider the removal of a file a change.

changed_on_remove
bool – A boolean value indicating if removed files count as a change.

files
set – The `set` of file names (relative to the current directory) that are being watched.

get_changed_files()
Get a `set` containing the changed files since the last call.

This method determines which files have changed since the last time this method was called. Added files, changed files (determined by the type of watcher) and removed files (if *changed_on_remove* is True) are all counted as changed files.

The watchers are stored and managed internally to this class.

Returns A `set` of files that have changed since the last time this method was called.

Return type set

glob_manager
GlobManager – The instance of the *GlobManager* that was passed in.

reglob
bool – A boolean value indicating whether we are re-evaluating file globs each time *get_changed_files()* is called.

watcher
Watcher – A reference to the *Watcher* class that is being used to detect changes.

3.1.5 Glob Manager

The *GlobManager* is responsible for expanding globs and ensuring that **only files** are returned.

Multiple globs can be passed in to the class, these are then expanded and matching files (no directories) are returned.

As an example, the following code would set up a *GlobManager* class that would find all files ending in `.py`.

```
>>> manager = GlobManager(['**/*.py'])
```

To actually get the files matching the specified globs the `get_files()` method can be called:

```
>>> manager.get_files()
```

class watch_do.glob_manager.**GlobManager**(*globs*)
This class expands the globs that are provided to it.

Multiple globs can be specified in order to watch a multitude of files.

Initialise the *GlobManager*.

Parameters **globs** (*list*) – A list of globs (as strings) that this class will expand.

get_files()
Expand the globs and return a set of matching files.

Returns A set of strings containing the files that matched the globs passed into this class.

Return type set

globs
set – A set of globs that were passed into this class.

last_files
set – The set of files last returned by the *get_files()* method.

3.1.6 Banner Builder

The *BannerBuilder* class is responsible for creating the banners that are displayed when using the command line interface to Watch Do.

Headers and footers are created in a format defined within this class that make use of the metadata that is passed into the build_header() and build_footer() methods.

As an example, the following code would return a header populated with the required metadata.

```
>>> BannerBuilder.build_header(
...     {'file1', 'file2'}, ModificationTime)
```

class watch_do.banner_builder.**BannerBuilder**
This class creates the headers and footers (banners) containing metrics that are used predominantly by the command line interface.

static build_footer(*trigger_time*, *trigger_cause*, *doer_run_time*, *files*, *watch_method*)
Build the footer from the provided metadata.

This interpolates the metadata provided by the parameters into a predefined string that can be used as information to display **after** a change has occurred.

Parameters

- **trigger_time** (*int*) – A timestamp of when the doers were triggered.
- **trigger_cause** (*list*) – A list of items that caused the doers to run. This list is joined with ‘, ‘ and the second to last and last item joined with ‘ and ‘.
- **doer_run_time** (*double*) – The duration of time the doers took to run.
- **files** (*set*) – A set containing the files that are currently being watched.
- **watch_method** (*Watcher*) – A reference to the class that is being used to watch the files.

Returns A string containing the generated footer.

Return type str

static **build_header**(*files*, *watch_method*)
Build a header from the provided metadata.

This interpolates the metadata provided by the parameters into a predefined string that can be used as information to display **before** a change has occurred.

Parameters

- **files** (*set*) – A `set` containing the files that are currently being watched.
- **watch_method** (*Watcher*) – A reference to the class that is being used to watch the files.

Returns A string containing the generated header.

Return type str

3.1.7 Exceptions

The exceptions defined in this module are used within the Watch Do package for reporting different error conditions.

None of the exceptions contain any extra logic, data or functionality, they only to provide a means to handle specific types of error.

exception watch_do.exceptions.**UnknownDoer**
This can be raised when a *Doer* cannot be found.

exception watch_do.exceptions.**UnknownWatcher**
This can be raised when a *Watcher* cannot be found.

CHAPTER 4

Indices and Tables

- genindex
- modindex
- search

Python Module Index

W

Index

Symbols

B

C

D

F

G

H

L

M

R

S

U

W