

---

# **Furtive Documentation**

***Release master***

**Derrick Bryant**

December 28, 2015



<b>1</b>	<b>Contents</b>	<b>3</b>
1.1	Furtive . . . . .	3
1.2	CLI Reference . . . . .	59
1.3	API Reference . . . . .	60
<b>2</b>	<b>Requirements</b>	<b>63</b>
<b>3</b>	<b>Getting Started</b>	<b>65</b>
<b>4</b>	<b>CLI Usage</b>	<b>67</b>
4.1	Use Case Example . . . . .	67
4.2	Actions . . . . .	67
<b>5</b>	<b>Tests</b>	<b>69</b>
<b>6</b>	<b>Faster YAML</b>	<b>71</b>
	<b>Python Module Index</b>	<b>73</b>



File Integrity Verification (furtive) aims to ensure long term data integrity verification for digital archival purposes. The idea is to create a manifest, or hash list, of all the files of which you wish to confirm integrity. Once a manifest has been created, a user can then confirm the integrity of files at any point in the future.

The documentation is available on Read The Docs at [furtive.readthedocs.org](https://furtive.readthedocs.org)



---

## Contents

---

### 1.1 Furtive

File Integrity Verification (furtive) aims to ensure long term data integrity verification for digital archival purposes. The idea is to create a manifest, or hash list, of all the files of which you wish to confirm integrity. Once a manifest has been created, a user can then confirm the integrity of files at any point in the future.

The documentation is available on Read The Docs at [furtive.readthedocs.org](https://furtive.readthedocs.org)

#### 1.1.1 Contents

##### Furtive

File Integrity Verification (furtive) aims to ensure long term data integrity verification for digital archival purposes. The idea is to create a manifest, or hash list, of all the files of which you wish to confirm integrity. Once a manifest has been created, a user can then confirm the integrity of files at any point in the future.

The documentation is available on Read The Docs at [furtive.readthedocs.org](https://furtive.readthedocs.org)

##### Contents

**Furtive** File Integrity Verification (furtive) aims to ensure long term data integrity verification for digital archival purposes. The idea is to create a manifest, or hash list, of all the files of which you wish to confirm integrity. Once a manifest has been created, a user can then confirm the integrity of files at any point in the future.

The documentation is available on Read The Docs at [furtive.readthedocs.org](https://furtive.readthedocs.org)

##### Contents

**Furtive** File Integrity Verification (furtive) aims to ensure long term data integrity verification for digital archival purposes. The idea is to create a manifest, or hash list, of all the files of which you wish to confirm integrity. Once a manifest has been created, a user can then confirm the integrity of files at any point in the future.

The documentation is available on Read The Docs at [furtive.readthedocs.org](https://furtive.readthedocs.org)

##### Contents

**Furtive** File Integrity Verification (furtive) aims to ensure long term data integrity verification for digital archival purposes. The idea is to create a manifest, or hash list, of all the files of which you wish to confirm integrity. Once a manifest has been created, a user can then confirm the integrity of files at any point in the future.

The documentation is available on Read The Docs at [furtive.readthedocs.org](https://furtive.readthedocs.org)

### Contents

**Furtive** File Integrity Verification (furtive) aims to ensure long term data integrity verification for digital archival purposes. The idea is to create a manifest, or hash list, of all the files of which you wish to confirm integrity. Once a manifest has been created, a user can then confirm the integrity of files at any point in the future.

The documentation is available on Read The Docs at [furtive.readthedocs.org](https://furtive.readthedocs.org)

### Contents

**Furtive** File Integrity Verification (furtive) aims to ensure long term data integrity verification for digital archival purposes. The idea is to create a manifest, or hash list, of all the files of which you wish to confirm integrity. Once a manifest has been created, a user can then confirm the integrity of files at any point in the future.

The documentation is available on Read The Docs at [furtive.readthedocs.org](https://furtive.readthedocs.org)

### Contents

**Furtive** File Integrity Verification (furtive) aims to ensure long term data integrity verification for digital archival purposes. The idea is to create a manifest, or hash list, of all the files of which you wish to confirm integrity. Once a manifest has been created, a user can then confirm the integrity of files at any point in the future.

The documentation is available on Read The Docs at [furtive.readthedocs.org](https://furtive.readthedocs.org)

### Contents

**Furtive** File Integrity Verification (furtive) aims to ensure long term data integrity verification for digital archival purposes. The idea is to create a manifest, or hash list, of all the files of which you wish to confirm integrity. Once a manifest has been created, a user can then confirm the integrity of files at any point in the future.

The documentation is available on Read The Docs at [furtive.readthedocs.org](https://furtive.readthedocs.org)

### Contents

**Furtive** File Integrity Verification (furtive) aims to ensure long term data integrity verification for digital archival purposes. The idea is to create a manifest, or hash list, of all the files of which you wish to confirm integrity. Once a manifest has been created, a user can then confirm the integrity of files at any point in the future.

The documentation is available on Read The Docs at [furtive.readthedocs.org](https://furtive.readthedocs.org)

### Contents



**Furtive** File Integrity Verification (furtive) aims to ensure long term data integrity verification for digital archival purposes. The idea is to create a manifest, or hash list, of all the files of which you wish to confirm integrity. Once a manifest has been created, a user can then confirm the integrity of files at any point in the future.

The documentation is available on Read The Docs at [furtive.readthedocs.org](http://furtive.readthedocs.org)

## Contents

**Furtive** File Integrity Verification (furtive) aims to ensure long term data integrity verification for digital archival purposes. The idea is to create a manifest, or hash list, of all the files of which you wish to confirm integrity. Once a manifest has been created, a user can then confirm the integrity of files at any point in the future.

The documentation is available on Read The Docs at [furtive.readthedocs.org](http://furtive.readthedocs.org)

## Contents

**CLI Reference** Command Line Interface (or Tool) reference.

Manage a Furtive manifest

```
usage: furtive [-h] [--basedir BASEDIR] [--manifest MANIFEST_PATH]
              [--log-level {debug,info,warn,error,critical}]
              [--exclude PATTERN] [--quiet] [--report-output FILE_NAME]
              [--version]
              {create,compare,check}
```

### Positional arguments:

<b>action</b>	Which action to perform: compare - compare the current state of the files on the file system with the recorded state in the manifest file. Status code is 0 if the comparison was successful. check - check the integrity of files listed in the manifest. Same as compare but exits with status code 1 if there are changes to the files included in the manifest. That is, if any file hash changes or if files are added or removed, the application will exit with a status code of 1 to indicate there are changes. This action can be useful for scripting. For example, to run a nightly cron check of a manifest. create - create a new manifest from the files in the directory specified by the --basedir argument.
---------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Possible choices: create, compare, check

### Options:

<b>--basedir=.</b>	Directory containing files that will be checked. Default: .
<b>--manifest</b>	Location of the manifest file. Manifests may be located outside the directory indicated by --basedir. Must provide path and filename of the manifest file. Default: <basedir>/manifest.yaml
<b>--log-level=info</b>	verbosity of furtive  Possible choices: debug, info, warn, error, critical
<b>--exclude=[]</b>	Patterns to exclude files and directories from manifest. Can have multiple occurrences of this argument. Excludes are not stored in the manifest so it is up to the user to provide the same arguments every run. Patterns are

evaluated as UNIX shell-style wildcard characters. See the [fnmatch documentation](https://docs.python.org/2/library/fnmatch.html) for more information.

It is important to note that exclusions are not stored. Therefore, they must be specified for every run of 'furtive'. Otherwise, the files which were previously excluded will be included and will show up as files added to the manifest.

<b>--quiet=False</b>	Only print out critical error messages. Do not print a report at the end of a compare run. Using this argument will override the log-level and set it to "critical". Only exceptions will be printed to terminal. The return code will be the only way to know if a Manifest has changed. This is useful for scripting such as a cron based manifest checks. Useful with the check command.
<b>--report-output=-</b>	File to print the diff report to. - for stdout. This can be consumed by other scripts to determine exactly what has changed within the manifest. Default: -
<b>--version</b>	show program's version number and exit

## API Reference

- *API Reference*
  - *Furtive Class*
  - *Sub-Modules*
    - \* *Hasher*
    - \* *Manifest*

This document is for developers of furtive, it contains the API functions

## Furtive Class Furtive - File Integrity Verification System

**class** `furtive.Furtive` (*base\_dir*, *manifest\_path*, *exclude=None*)  
Bases: `object`

Furtive is an application which stores file state and allows users to verify the state in the future. Example use cases include file archives and file transport.

If the manifest file exists, it will be automatically loaded. Calling `create()` will overwrite the existing manifest in memory as well as the file.

### Parameters

- **base\_dir** (*str*) – Base directory to use for the manifest. Can be a full or relative path.
- **manifest\_path** (*str*) – Path to the manifest file. Can be a full or relative path.
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

### `compare()`

Compare the hashes in the database with the current hashes of files on the file system.

**Returns** Dictionary of added, deleted, and changed files.

**Return type** dict

**create()**

Create and save a new manifest.

The contents of the new Manifest() will be saved to *manifest\_path*.

**Returns** None

**Sub-Modules**

**Hasher** Manages the hashing of files

**class** furtive.hasher.**HashDirectory** (*directory*, *exclude=None*)

Bases: object

Object to manage hashing files in a directory.

This object is responsible for walking the directory tree and adding each file to a list. Once the directory walk has completed, each file path is passed to `hash_task()`. After each file has been hashed, this object will then create a Python dictionary of files with their associated hash.

**Parameters**

- **directory** (*str*) – Path to directory containing files
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

**Returns** Dictionary of file:hash

**Return type** dict

**excluded** (*file\_path*)

Should the file be excluded from the manifest?

Determines if a file should be excluded based on UNIX style pattern matching. Think `*`, `?`, and `[]` sequences.

For matchers, see <https://docs.python.org/2/library/fnmatch.html>

**Parameters** **file\_path** (*str*) – path of the file to match against.

**Returns** True or False indicating if the file should be excluded from the list of files contained within the manifest.

**Return type** bool

**hash\_files()**

Orchestrates the discovery and hashing of files.

Note: This method only supports the md5 hashing algorithm

**furtive.hasher.hash\_task** (*file\_path*, *hash\_algorithm='md5'*)

Responsible for hashing a file.

This function reads in the file *file\_path* in small chunks the size of the hash algorithm's block size in order to avoid running out of memory. This means that this function should be able to read any file regardless of the size.

**Parameters**

- **file\_path** (*str*) – path of file to hash
- **hash\_algorithm** (*str*) – the hashing algorithm to use. All options available in *hashlib* should work. See: <https://docs.python.org/2/library/hashlib.html>

**Returns** hash of file

**Return type** dict

`furtive.hasher.initializer(terminating_)`

Method to make terminating a global variable so that it is inherited by child processes.

**Manifest** Manifest of files and their hashes

**class** `furtive.manifest.Manifest(directory, manifest_file, exclude=None)`

Bases: object

Manifest of files and the associated hashes.

**Parameters**

- **directory** – directory which will serve as the root for the manifest. All files under the directory will be hashed and added to or compared with the manifest.
- **type** – str
- **manifest\_file** – file location of the manifest file. This is the path which will be used for the `create()` and `compare()` methods. If the file exists, the `create()` method will overwrite it.
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

**create()**

Creates a new manifest from the directory by calling `furtive.hasher.HashDirectory()` and placing the return dictionary in to *Manifest.manifest*.

**is\_empty()**

Determines if the manifest within memory is empty.

This simply checks to see if the manifest is None.

**Returns** True if manifest is empty, False otherwise.

**Return type** bool

**load()**

Load a manifest from the manifest file.

This method will open the manifest YAML file and load it in to the *manifest* object variable.

**save()**

Save the manifest to the manifest file.

Open a YAML file and dump the contents of the manifest to it.

**Requirements**

- Python 2.7, 3.4, or 3.5

**Getting Started** To install furtive, run `pip install furtive`.

**CLI Usage** See the [CLI Reference](#) for more information about available command line arguments.

**Use Case Example** Suppose you have a million digital photos in a directory called `my-photos` that you have taken over the years. You would like to know if the files begin to decay due to hardware failure or something else. Alternatively, you may wish to have reassurance that your photos have not become corrupted while being stored in a cloud backup solution such as S3 or Glacier.

To record the current state of the files, run `furtive --basedir my-photos create`

This command creates the file `.manifest.yaml` in the `my-photos/` directory. The location and name of this file can be changed by using the `--manifest` argument.

At this point, you can be sure that you will know if a file has changed. To check the files on the file system to the recorded state in the manifest, run `furtive --basedir my-photos check`. The application will output a list of files which have been added, removed, or changed. This output is YAML format so it should be easy to parse. Additionally, `furtive` will exit with 1 indicating the check failed. This command can be placed in a cron job and setup to send a notification if a file has changed.

**Actions** There are a few actions which can be performed by `furtive`.

- **create** - create a new manifest from the files in the directory specified by the `--basedir` argument.
- **compare** - compare the current state of the files on the file system with the recorded state in the manifest file. A YAML based report will be created detailing which files have changed and which files have been added or removed. Status code is 0 if the comparison was successful.
- **check** - check the integrity of files listed in the manifest. Same as `compare` but exits with status code 1 if there are changes to the files included in the manifest. That is, if any file hash changes or if files are added or removed, the application will exit with a status code of 1 to indicate there are changes. This action can be useful for scripting. For example, to run a nightly cron check of a manifest. A YAML based report will be generated as well.

**Tests** This application comes with tests. To run them, ensure you have `tox` installed (`pip install tox`). Then you can run `tox` to run the tests.

To build the docs, run `tox -e docs`. The HTML docs will be generated in the `.tox/docs/tmp/html/` directory.

**Faster YAML** By default, `furtive` will install and use the full Python implementation of the YAML parser which is very slow. In a testing environment, the Python implementation of the YAML loader took 1 minute to parse a 187,000 line `furtive` manifest file. By contrast, when the [LibYaml](#) parser was used, the loader took only 5 seconds to parse the same file.

To install the faster parser, perform the following steps:

1. Follow the [instructions from the LibYaml website](#) to download and install the latest release of `libyaml`.
2. Reinstall the `PyYAML` package by downloading the latest tar from the [PyYAML website](#) and running `python setup.py --with-libyaml install`

**CLI Reference** Command Line Interface (or Tool) reference.

Manage a Furtive manifest

```
usage: furtive [-h] [--basedir BASEDIR] [--manifest MANIFEST_PATH]
              [--log-level {debug,info,warn,error,critical}]
              [--exclude PATTERN] [--quiet] [--report-output FILE_NAME]
              [--version]
              {create,compare,check}
```

## Positional arguments:

**action** Which action to perform: compare - compare the current state of the files on the file system with the recorded state in the manifest file. Status code is 0 if the comparison was successful. check - check the integrity of files listed in the manifest. Same as compare but exits with status code 1 if there are changes to the files included in the manifest. That is, if any file hash changes or if files are added or removed, the application will exit with a status code of 1 to indicate there are changes. This action can be useful for scripting. For example, to run a nightly cron check of a manifest. create - create a new manifest from the files in the directory specified by the --basedir argument.

Possible choices: create, compare, check

## Options:

**--basedir=.** Directory containing files that will be checked. Default: .

**--manifest** Location of the manifest file. Manifests may be located outside the directory indicated by --basedir. Must provide path and filename of the manifest file. Default: <basedir>/manifest.yaml

**--log-level=info** verbosity of furtive

Possible choices: debug, info, warn, error, critical

**--exclude=[]** Patterns to exclude files and directories from manifest. Can have multiple occurrences of this argument. Excludes are not stored in the manifest so it is up to the user to provide the same arguments every run. Patterns are evaluated as UNIX shell-style wildcard characters. See the [fnmatch documentation](https://docs.python.org/2/library/fnmatch.html) for more information.

It is important to note that exclusions are not stored. Therefore, they must be specified for every run of 'furtive'. Otherwise, the files which were previously excluded will be included and will show up as files added to the manifest.

**--quiet=False** Only print out critical error messages. Do not print a report at the end of a compare run. Using this argument will override the log-level and set it to "critical". Only exceptions will be printed to terminal. The return code will be the only way to know if a Manifest has changed. This is useful for scripting such as a cron based manifest checks. Useful with the check command.

**--report-output=-** File to print the diff report to. - for stdout. This can be consumed by other scripts to determine exactly what has changed within the manifest. Default: -

**--version** show program's version number and exit

## API Reference

- *API Reference*
  - *Furtive Class*
  - *Sub-Modules*
    - \* *Hasher*
    - \* *Manifest*

This document is for developers of furtive, it contains the API functions

## **Furtive Class** Furtive - File Integrity Verification System

**class** furtive.**Furtive** (*base\_dir*, *manifest\_path*, *exclude=None*)

Bases: object

Furtive is an application which stores file state and allows users to verify the state in the future. Example use cases include file archives and file transport.

If the manifest file exists, it will be automatically loaded. Calling create() will overwrite the existing manifest in memory as well as the file.

### **Parameters**

- **base\_dir** (*str*) – Base directory to use for the manifest. Can be a full or relative path.
- **manifest\_path** (*str*) – Path to the manifest file. Can be a full or relative path.
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

**compare** ()

Compare the hashes in the database with the current hashes of files on the file system.

**Returns** Dictionary of added, deleted, and changed files.

**Return type** dict

**create** ()

Create and save a new manifest.

The contents of the new Manifest() will be saved to *manifest\_path*.

**Returns** None

## **Sub-Modules**

**Hasher** Manages the hashing of files

**class** furtive.hasher.**HashDirectory** (*directory*, *exclude=None*)

Bases: object

Object to manage hashing files in a directory.

This object is responsible for walking the directory tree and adding each file to a list. Once the directory walk has completed, each file path is passed to hash\_task(). After each file has been hashed, this object will then create a Python dictionary of files with their associated hash.

### **Parameters**

- **directory** (*str*) – Path to directory containing files
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

**Returns** Dictionary of file:hash

**Return type** dict

**excluded** (*file\_path*)

Should the file be excluded from the manifest?

Determines if a file should be excluded based on UNIX style pattern matching. Think \*, ?, and [] sequences.

For matchers, see <https://docs.python.org/2/library/fnmatch.html>

**Parameters** `file_path` (*str*) – path of the file to match against.

**Returns** True or False indicating if the file should be excluded from the list of files contained within the manifest.

**Return type** bool

**hash\_files()**

Orchestrates the discovery and hashing of files.

Note: This method only supports the md5 hashing algorithm

`furtive.hasher.hash_task(file_path, hash_algorithm='md5')`

Responsible for hashing a file.

This function reads in the file `file_path` in small chunks the size of the hash algorithm's block size in order to avoid running out of memory. This means that this function should be able to read any file regardless of the size.

**Parameters**

- **file\_path** (*str*) – path of file to hash
- **hash\_algorithm** (*str*) – the hashing algorithm to use. All options available in *hashlib.algorithms* should work. See: <https://docs.python.org/2/library/hashlib.html>

**Returns** hash of file

**Return type** dict

`furtive.hasher.initializer(terminating_)`

Method to make terminating a global variable so that it is inherited by child processes.

**Manifest** Manifest of files and their hashes

**class** `furtive.manifest.Manifest(directory, manifest_file, exclude=None)`

Bases: object

Manifest of files and the associated hashes.

**Parameters**

- **directory** – directory which will serve as the root for the manifest. All files under the directory will be hashed and added to or compared with the manifest.
- **type** – str
- **manifest\_file** – file location of the manifest file. This is the path which will be used for the `create()` and `compare()` methods. If the file exists, the `create()` method will overwrite it.
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

**create()**

Creates a new manifest from the directory by calling `furtive.hasher.HashDirectory()` and placing the return dictionary in to *Manifest.manifest*.

**is\_empty()**

Determines if the manifest within memory is empty.

This simply checks to see if the manifest is None.

**Returns** True if manifest is empty, False otherwise.



**Return type** bool

**load()**

Load a manifest from the manifest file.

This method will open the manifest YAML file and load it in to the *manifest* object variable.

**save()**

Save the manifest to the manifest file.

Open a YAML file and dump the contents of the manifest to it.

## Requirements

- Python 2.7, 3.4, or 3.5

**Getting Started** To install furtive, run `pip install furtive`.

**CLI Usage** See the [CLI Reference](#) for more information about available command line arguments.

**Use Case Example** Suppose you have a million digital photos in a directory called `my-photos` that you have taken over the years. You would like to know if the files begin to decay due to hardware failure or something else. Alternatively, you may wish to have reassurance that your photos have not become corrupted while being stored in a cloud backup solution such as S3 or Glacier.

To record the current state of the files, run `furtive --basedir my-photos create`

This command creates the file `.manifest.yaml` in the `my-photos/` directory. The location and name of this file can be changed by using the `--manifest` argument.

At this point, you can be sure that you will know if a file has changed. To check the files on the file system to the recorded state in the manifest, run `furtive --basedir my-photos check`. The application will output a list of files which have been added, removed, or changed. This output is YAML format so it should be easy to parse. Additionally, furtive will exit with 1 indicating the check failed. This command can be placed in a cron job and setup to send a notification if a file has changed.

**Actions** There are a few actions which can be performed by furtive.

- **create** - create a new manifest from the files in the directory specified by the `--basedir` argument.
- **compare** - compare the current state of the files on the file system with the recorded state in the manifest file. A YAML based report will be created detailing which files have changed and which files have been added or removed. Status code is 0 if the comparison was successful.
- **check** - check the integrity of files listed in the manifest. Same as `compare` but exits with status code 1 if there are changes to the files included in the manifest. That is, if any file hash changes or if files are added or removed, the application will exit with a status code of 1 to indicate there are changes. This action can be useful for scripting. For example, to run a nightly cron check of a manifest. A YAML based report will be generated as well.

**Tests** This application comes with tests. To run them, ensure you have `tox` installed (`pip install tox`). Then you can run `tox` to run the tests.

To build the docs, run `tox -e docs`. The HTML docs will be generated in the `.tox/docs/tmp/html/` directory.

**Faster YAML** By default, furtive will install and use the full Python implementation of the YAML parser which is very slow. In a testing environment, the Python implementation of the YAML loader took 1 minute to parse a 187,000 line furtive manifest file. By contrast, when the [LibYaml](#) parser was used, the loader took only 5 seconds to parse the same file.

To install the faster parser, perform the following steps:

1. Follow the [instructions from the LibYaml website](#) to download and install the latest release of libyaml.
2. Reinstall the PyYAML package by downloading the latest tar from the [PyYAML website](#) and running `python setup.py --with-libyaml install`

**CLI Reference** Command Line Interface (or Tool) reference.

Manage a Furtive manifest

```
usage: furtive [-h] [--basedir BASEDIR] [--manifest MANIFEST_PATH]
              [--log-level {debug,info,warn,error,critical}]
              [--exclude PATTERN] [--quiet] [--report-output FILE_NAME]
              [--version]
              {create,compare,check}
```

**Positional arguments:**

<b>action</b>	Which action to perform: compare - compare the current state of the files on the file system with the recorded state in the manifest file. Status code is 0 if the comparison was successful. check - check the integrity of files listed in the manifest. Same as compare but exits with status code 1 if there are changes to the files included in the manifest. That is, if any file hash changes or if files are added or removed, the application will exit with a status code of 1 to indicate there are changes. This action can be useful for scripting. For example, to run a nightly cron check of a manifest. create - create a new manifest from the files in the directory specified by the --basedir argument.
---------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Possible choices: create, compare, check

**Options:**

<b>--basedir=.</b>	Directory containing files that will be checked. Default: .
<b>--manifest</b>	Location of the manifest file. Manifests may be located outside the directory indicated by --basedir. Must provide path and filename of the manifest file. Default: <basedir>/manifest.yaml
<b>--log-level=info</b>	verbosity of furtive Possible choices: debug, info, warn, error, critical
<b>--exclude=[]</b>	Patterns to exclude files and directories from manifest. Can have multiple occurrences of this argument. Excludes are not stored in the manifest so it is up to the user to provide the same arguments every run. Patterns are evaluated as UNIX shell-style wildcard characters. See the <a href="https://docs.python.org/2/library/fnmatch.html">[fnmatch documentation](https://docs.python.org/2/library/fnmatch.html)</a> for more information.

It is important to note that exclusions are not stored. Therefore, they must be specified for every run of 'furtive'. Otherwise, the files which were previously excluded will be included and will show up as files added to the manifest.

<b>--quiet=False</b>	Only print out critical error messages. Do not print a report at the end of a compare run. Using this argument will override the log-level and set it to “critical”. Only acceptions will be printed to terminal. The return code will be the only way to know if a Manifest has changed. This is useful for scripting such as a cron based manifest checks. Useful with the check command.
<b>--report-output=-</b>	File to print the diff report to. - for stdout.This can be consumed by other scripts todetermine exactly what has changed within themanifestDefault: -
<b>--version</b>	show program’s version number and exit

## API Reference

- *API Reference*
  - *Furtive Class*
  - *Sub-Modules*
    - \* *Hasher*
    - \* *Manifest*

This document is for developers of furtive, it contains the API functions

## Furtive Class Furtive - File Integrity Verification System

**class** `furtive.Furtive` (*base\_dir*, *manifest\_path*, *exclude=None*)

Bases: `object`

Furtive is an application which stores file state and allows users to verify the state in the future. Example use cases include file archives and file transport.

If the manifest file exists, it will be automatcally loaded. Calling `create()` will overwrite the existing manifest in memory as well as the file.

### Parameters

- **base\_dir** (*str*) – Base directory to use for the manifest. Can be a full or relative path.
- **manifest\_path** (*str*) – Path to the manifest file. Can be a full or relative path.
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

### `compare()`

Compare the hashes in the database with the current hashes of files on the file system.

**Returns** Dictionary of added, deleted, and changed files.

**Return type** `dict`

### `create()`

Create and save a new manifest.

The contents of the new Manfiest() will be saved to *manifest\_path*.

**Returns** `None`

## Sub-Modules

**Hasher** Manages the hashing of files

**class** `furtive.hasher.HashDirectory(directory, exclude=None)`

Bases: `object`

Object to manage hashing files in a directory.

This object is responsible for walking the directory tree and adding each file to a list. Once the directory walk has completed, each file path is passed to `hash_task()`. After each file has been hashed, this object will then create a Python dictionary of files with their associated hash.

**Parameters**

- **directory** (*str*) – Path to directory containing files
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

**Returns** Dictionary of file:hash

**Return type** `dict`

**excluded** (*file\_path*)

Should the file be excluded from the manifest?

Determines if a file should be excluded based on UNIX style pattern matching. Think `*`, `?`, and `[]` sequences.

For matchers, see <https://docs.python.org/2/library/fnmatch.html>

**Parameters** **file\_path** (*str*) – path of the file to match against.

**Returns** True or False indicating if the file should be excluded from the list of files contained within the manifest.

**Return type** `bool`

**hash\_files** ()

Orchestrates the discovery and hashing of files.

Note: This method only supports the md5 hashing algorithm

**furtive.hasher.hash\_task** (*file\_path*, *hash\_algorithm*='md5')

Responsible for hashing a file.

This function reads in the file `file_path` in small chunks the size of the hash algorithm's block size in order to avoid running out of memory. This means that this function should be able to read any file irregardless of the size.

**Parameters**

- **file\_path** (*str*) – path of file to hash
- **hash\_algorithm** (*str*) – the hashing algorithm to use. All options available in `hashlib.algorithms` should work. See: <https://docs.python.org/2/library/hashlib.html>

**Returns** hash of file

**Return type** `dict`

**furtive.hasher.initializer** (*terminating\_*)

Method to make terminating a global variable so that it is inherited by child processes.

**Manifest** Manifest of files and their hashes

**class** `furtive.manifest.Manifest` (*directory*, *manifest\_file*, *exclude=None*)

Bases: `object`

Manifest of files and the associated hashes.

#### Parameters

- **directory** – directory which will serve as the root for the manifest. All files under the directory will be hashed and added to or compared with the manifest.
- **type** – str
- **manifest\_file** – file location of the manifest file. This is the path which will be used for the `create()` and `compare()` methods. If the file exists, the `create()` method will overwrite it.
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

#### `create()`

Creates a new manifest from the directory by calling `furtive.hasher.HashDirectory()` and placing the return dictionary in to *Manifest.manifest*.

#### `is_empty()`

Determines if the manifest within memory is empty.

This simply checks to see if the manifest is `None`.

**Returns** True if manifest is empty, False otherwise.

**Return type** bool

#### `load()`

Load a manifest from the manifest file.

This method will open the manifest YAML file and load it in to the *manifest* object variable.

#### `save()`

Save the manifest to the manifest file.

Open a YAML file and dump the contents of the manifest to it.

## Requirements

- Python 2.7, 3.4, or 3.5

**Getting Started** To install furtive, run `pip install furtive`.

**CLI Usage** See the [CLI Reference](#) for more information about available command line arguments.

**Use Case Example** Suppose you have a million digital photos in a directory called `my-photos` that you have taken over the years. You would like to know if the files begin to decay due to hardware failure or something else. Alternatively, you may wish to have reassurance that your photos have not become corrupted while being stored in a cloud backup solution such as S3 or Glacier.

To record the current state of the files, run `furtive --basedir my-photos create`

This command creates the file `.manifest.yaml` in the `my-photos/` directory. The location and name of this file can be changed by using the `--manifest` argument.

At this point, you can be sure that you will know if a file has changed. To check the files on the file system to the recorded state in the manifest, run `furtive --basedir my-photos check`. The application will output a list of files which have been added, removed, or changed. This output is YAML format so it should be easy to parse. Additionally, `furtive` will exit with 1 indicating the check failed. This command can be placed in a cron job and setup to send a notification if a file has changed.

**Actions** There are a few actions which can be performed by `furtive`.

- **create** - create a new manifest from the files in the directory specified by the `--basedir` argument.
- **compare** - compare the current state of the files on the file system with the recorded state in the manifest file. A YAML based report will be created detailing which files have changed and which files have been added or removed. Status code is 0 if the comparison was successful.
- **check** - check the integrity of files listed in the manifest. Same as `compare` but exits with status code 1 if there are changes to the files included in the manifest. That is, if any file hash changes or if files are added or removed, the application will exit with a status code of 1 to indicate there are changes. This action can be useful for scripting. For example, to run a nightly cron check of a manifest. A YAML based report will be generated as well.

**Tests** This application comes with tests. To run them, ensure you have `tox` installed (`pip install tox`). Then you can run `tox` to run the tests.

To build the docs, run `tox -e docs`. The HTML docs will be generated in the `.tox/docs/tmp/html/` directory.

**Faster YAML** By default, `furtive` will install and use the full Python implementation of the YAML parser which is very slow. In a testing environment, the Python implementation of the YAML loader took 1 minute to parse a 187,000 line `furtive` manifest file. By contrast, when the [LibYaml](#) parser was used, the loader took only 5 seconds to parse the same file.

To install the faster parser, perform the following steps:

1. Follow the [instructions from the LibYaml website](#) to download and install the latest release of `libyaml`.
2. Reinstall the `PyYAML` package by downloading the latest tar from the [PyYAML website](#) and running `python setup.py --with-libyaml install`

**CLI Reference** Command Line Interface (or Tool) reference.

Manage a Furtive manifest

```
usage: furtive [-h] [--basedir BASEDIR] [--manifest MANIFEST_PATH]
              [--log-level {debug,info,warn,error,critical}]
              [--exclude PATTERN] [--quiet] [--report-output FILE_NAME]
              [--version]
              {create,compare,check}
```

**Positional arguments:**

action	
	Which action to perform: <code>compare</code> - compare the current state of the files on the file system with the recorded state in the manifest file. Status code is 0 if the comparison was successful. <code>check</code> - check the integrity of files listed in the manifest. Same as <code>compare</code> but exits with status code 1 if there are changes to the files included in the manifest. That is, if any file hash changes or if files are added or removed, the application will exit with a status code of 1 to indicate there are changes. This action can be useful for

scripting. For example, to run a nightly cron check of a manifest. create  
- create a new manifest from the files in the directory specified by the --basedir argument.

Possible choices: create, compare, check

#### Options:

<b>--basedir=.</b>	Directory containing files that will be checked. Default: .
<b>--manifest</b>	Location of the manifest file. Manifests may be located outside the directory indicated by --basedir. Must provide path and filename of the manifest file. Default: <basedir>/manifest.yaml
<b>--log-level=info</b>	verbosity of furtive  Possible choices: debug, info, warn, error, critical
<b>--exclude=[]</b>	Patterns to exclude files and directories from manifest. Can have multiple occurrences of this argument. Excludes are not stored in the manifest so it is up to the user to provide the same arguments every run. Patterns are evaluated as UNIX shell-style wildcard characters. See the [fnmatch documentation](https://docs.python.org/2/library/fnmatch.html) for more information.  It is important to note that exclusions are not stored. Therefore, they must be specified for every run of 'furtive'. Otherwise, the files which were previously excluded will be included and will show up as files added to the manifest.
<b>--quiet=False</b>	Only print out critical error messages. Do not print a report at the end of a compare run. Using this argument will override the log-level and set it to "critical". Only exceptions will be printed to terminal. The return code will be the only way to know if a Manifest has changed. This is useful for scripting such as a cron based manifest checks. Useful with the check command.
<b>--report-output=-</b>	File to print the diff report to. - for stdout. This can be consumed by other scripts to determine exactly what has changed within the manifest. Default: -
<b>--version</b>	show program's version number and exit

#### API Reference

- *API Reference*
  - *Furtive Class*
  - *Sub-Modules*
    - \* *Hasher*
    - \* *Manifest*

This document is for developers of furtive, it contains the API functions

**Furtive Class** Furtive - File Integrity Verification System

```
class furtive.Furtive(base_dir, manifest_path, exclude=None)
    Bases: object
```

Furtive is an application which stores file state and allows users to verify the state in the future. Example use cases include file archives and file transport.

If the manifest file exists, it will be automatically loaded. Calling `create()` will overwrite the existing manifest in memory as well as the file.

### Parameters

- **base\_dir** (*str*) – Base directory to use for the manifest. Can be a full or relative path.
- **manifest\_path** (*str*) – Path to the manifest file. Can be a full or relative path.
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

### `compare()`

Compare the hashes in the database with the current hashes of files on the file system.

**Returns** Dictionary of added, deleted, and changed files.

**Return type** dict

### `create()`

Create and save a new manifest.

The contents of the new Manifest() will be saved to *manifest\_path*.

**Returns** None

## Sub-Modules

**Hasher** Manages the hashing of files

**class** `furtive.hasher.HashDirectory` (*directory*, *exclude=None*)

Bases: object

Object to manage hashing files in a directory.

This object is responsible for walking the directory tree and adding each file to a list. Once the directory walk has completed, each file path is passed to `hash_task()`. After each file has been hashed, this object will then create a Python dictionary of files with their associated hash.

### Parameters

- **directory** (*str*) – Path to directory containing files
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

**Returns** Dictionary of file:hash

**Return type** dict

### `excluded` (*file\_path*)

Should the file be excluded from the manifest?

Determines if a file should be excluded based on UNIX style pattern matching. Think `*`, `?`, and `[]` sequences.

For matchers, see <https://docs.python.org/2/library/fnmatch.html>

**Parameters** **file\_path** (*str*) – path of the file to match against.

**Returns** True or False indicating if the file should be excluded from the list of files contained within the manifest.

**Return type** bool



**hash\_files()**

Orchestrates the discovery and hashing of files.

Note: This method only supports the md5 hashing algorithm

`furtive.hasher.hash_task(file_path, hash_algorithm='md5')`

Responsible for hashing a file.

This function reads in the file `file_path` in small chunks the size of the hash algorithm's block size in order to avoid running out of memory. This means that this function should be able to read any file regardless of the size.

**Parameters**

- **file\_path** (*str*) – path of file to hash
- **hash\_algorithm** (*str*) – the hashing algorithm to use. All options available in *hashlib.algorithms* should work. See: <https://docs.python.org/2/library/hashlib.html>

**Returns** hash of file

**Return type** dict

`furtive.hasher.initializer(terminating_)`

Method to make terminating a global variable so that it is inherited by child processes.

**Manifest** Manifest of files and their hashes

**class** `furtive.manifest.Manifest(directory, manifest_file, exclude=None)`

Bases: object

Manifest of files and the associated hashes.

**Parameters**

- **directory** – directory which will serve as the root for the manifest. All files under the directory will be hashed and added to or compared with the manifest.
- **type** – str
- **manifest\_file** – file location of the manifest file. This is the path which will be used for the `create()` and `compare()` methods. If the file exists, the `create()` method will overwrite it.
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

**create()**

Creates a new manifest from the directory by calling `furtive.hasher.HashDirectory()` and placing the return dictionary in to *Manifest.manifest*.

**is\_empty()**

Determines if the manifest within memory is empty.

This simply checks to see if the manifest is None.

**Returns** True if manifest is empty, False otherwise.

**Return type** bool

**load()**

Load a manifest from the manifest file.

This method will open the manifest YAML file and load it in to the *manifest* object variable.

### **save()**

Save the manifest to the manifest file.

Open a YAML file and dump the contents of the manifest to it.

## Requirements

- Python 2.7, 3.4, or 3.5

**Getting Started** To install furtive, run `pip install furtive`.

**CLI Usage** See the [CLI Reference](#) for more information about available command line arguments.

**Use Case Example** Suppose you have a million digital photos in a directory called `my-photos` that you have taken over the years. You would like to know if the files begin to decay due to hardware failure or something else. Alternatively, you may wish to have reassurance that your photos have not become corrupted while being stored in a cloud backup solution such as S3 or Glacier.

To record the current state of the files, run `furtive --basedir my-photos create`

This command creates the file `.manifest.yaml` in the `my-photos/` directory. The location and name of this file can be changed by using the `--manifest` argument.

At this point, you can be sure that you will know if a file has changed. To check the files on the file system to the recorded state in the manifest, run `furtive --basedir my-photos check`. The application will output a list of files which have been added, removed, or changed. This output is YAML format so it should be easy to parse. Additionally, furtive will exit with 1 indicating the check failed. This command can be placed in a cron job and setup to send a notification if a file has changed.

**Actions** There are a few actions which can be performed by furtive.

- **create** - create a new manifest from the files in the directory specified by the `--basedir` argument.
- **compare** - compare the current state of the files on the file system with the recorded state in the manifest file. A YAML based report will be created detailing which files have changed and which files have been added or removed. Status code is 0 if the comparison was successful.
- **check** - check the integrity of files listed in the manifest. Same as `compare` but exits with status code 1 if there are changes to the files included in the manifest. That is, if any file hash changes or if files are added or removed, the application will exit with a status code of 1 to indicate there are changes. This action can be useful for scripting. For example, to run a nightly cron check of a manifest. A YAML based report will be generated as well.

**Tests** This application comes with tests. To run them, ensure you have `tox` installed (`pip install tox`). Then you can run `tox` to run the tests.

To build the docs, run `tox -e docs`. The HTML docs will be generated in the `.tox/docs/tmp/html/` directory.

**Faster YAML** By default, furtive will install and use the full Python implementation of the YAML parser which is very slow. In a testing environment, the Python implementation of the YAML loader took 1 minute to parse a 187,000 line furtive manifest file. By contrast, when the [LibYaml](#) parser was used, the loader took only 5 seconds to parse the same file.

To install the faster parser, perform the following steps:

1. Follow the [instructions from the LibYaml website](#) to download and install the latest release of libyaml.
2. Reinstall the PyYAML package by downloading the latest tar from the [PyYAML website](#) and running `python setup.py --with-libyaml install`

## CLI Reference Command Line Interface (or Tool) reference.

### Manage a Furtive manifest

```
usage: furtive [-h] [--basedir BASEDIR] [--manifest MANIFEST_PATH]
              [--log-level {debug,info,warn,error,critical}]
              [--exclude PATTERN] [--quiet] [--report-output FILE_NAME]
              [--version]
              {create,compare,check}
```

### Positional arguments:

**action** Which action to perform: compare - compare the current state of the files on the file system with the recorded state in the manifest file. Status code is 0 if the comparison was successful. check - check the integrity of files listed in the manifest. Same as compare but exits with status code 1 if there are changes to the files included in the manifest. That is, if any file hash changes or if files are added or removed, the application will exit with a status code of 1 to indicate there are changes. This action can be useful for scripting. For example, to run a nightly cron check of a manifest. create - create a new manifest from the files in the directory specified by the --basedir argument.

Possible choices: create, compare, check

### Options:

**--basedir=.** Directory containing files that will be checked. Default: .

**--manifest** Location of the manifest file. Manifests may be located outside the directory indicated by --basedir. Must provide path and filename of the manifest file. Default: <basedir>/manifest.yaml

**--log-level=info** verbosity of furtive

Possible choices: debug, info, warn, error, critical

**--exclude=[]** Patterns to exclude files and directories from manifest. Can have multiple occurrences of this argument. Excludes are not stored in the manifest so it is up to the user to provide the same arguments every run. Patterns are evaluated as UNIX shell-style wildcard characters. See the [\[fnmatch documentation\]\(https://docs.python.org/2/library/fnmatch.html\)](https://docs.python.org/2/library/fnmatch.html) for more information.

It is important to note that exclusions are not stored. Therefore, they must be specified for every run of 'furtive'. Otherwise, the files which were previously excluded will be included and will show up as files added to the manifest.

**--quiet=False** Only print out critical error messages. Do not print a report at the end of a compare run. Using this argument will override the log-level and set it to "critical". Only exceptions will be printed to terminal. The return code will be the only way to know if a Manifest has changed. This is useful for scripting such as a cron based manifest checks. Useful with the check command.

**--report-output=-** File to print the diff report to. - for stdout. This can be consumed by other scripts to determine exactly what has changed within the manifest. Default: -

**--version** show program's version number and exit

### API Reference

- *API Reference*
  - *Furtive Class*
  - *Sub-Modules*
    - \* *Hasher*
    - \* *Manifest*

This document is for developers of furtive, it contains the API functions

### Furtive Class Furtive - File Integrity Verification System

**class** `furtive.Furtive` (*base\_dir*, *manifest\_path*, *exclude=None*)

Bases: `object`

Furtive is an application which stores file state and allows users to verify the state in the future. Example use cases include file archives and file transport.

If the manifest file exists, it will be automatically loaded. Calling `create()` will overwrite the existing manifest in memory as well as the file.

#### Parameters

- **base\_dir** (*str*) – Base directory to use for the manifest. Can be a full or relative path.
- **manifest\_path** (*str*) – Path to the manifest file. Can be a full or relative path.
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

**compare** ()

Compare the hashes in the database with the current hashes of files on the file system.

**Returns** Dictionary of added, deleted, and changed files.

**Return type** dict

**create** ()

Create and save a new manifest.

The contents of the new Manifest() will be saved to *manifest\_path*.

**Returns** None

### Sub-Modules

#### Hasher Manages the hashing of files

**class** `furtive.hasher.HashDirectory` (*directory*, *exclude=None*)

Bases: `object`

Object to manage hashing files in a directory.

This object is responsible for walking the directory tree and adding each file to a list. Once the directory walk has completed, each file path is passed to `hash_task()`. After each file has been hashed, this object will then create a Python dictionary of files with their associated hash.

#### Parameters

- **directory** (*str*) – Path to directory containing files
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

**Returns** Dictionary of file:hash

**Return type** dict

#### **excluded** (*file\_path*)

Should the file be excluded from the manifest?

Determines if a file should be excluded based on UNIX style pattern matching. Think `*`, `?`, and `[]` sequences.

For matchers, see <https://docs.python.org/2/library/fnmatch.html>

**Parameters** **file\_path** (*str*) – path of the file to match against.

**Returns** True or False indicating if the file should be excluded from the list of files contained within the manifest.

**Return type** bool

#### **hash\_files** ()

Orchestrates the discovery and hashing of files.

Note: This method only supports the md5 hashing algorithm

`furtive.hasher.hash_task(file_path, hash_algorithm='md5')`

Responsible for hashing a file.

This function reads in the file `file_path` in small chunks the size of the hash algorithm's block size in order to avoid running out of memory. This means that this function should be able to read any file regardless of the size.

#### Parameters

- **file\_path** (*str*) – path of file to hash
- **hash\_algorithm** (*str*) – the hashing algorithm to use. All options available in `hashlib.algorithms` should work. See: <https://docs.python.org/2/library/hashlib.html>

**Returns** hash of file

**Return type** dict

`furtive.hasher.initializer(terminating_)`

Method to make terminating a global variable so that it is inherited by child processes.

**Manifest** Manifest of files and their hashes

**class** `furtive.manifest.Manifest` (*directory, manifest\_file, exclude=None*)

Bases: object

Manifest of files and the associated hashes.

#### Parameters

- **directory** – directory which will serve as the root for the manifest. All files under the directory will be hashed and added to or compared with the manifest.

- **type** – str
- **manifest\_file** – file location of the manifest file. This is the path which will be used for the `create()` and `compare()` methods. If the file exists, the `create()` method will overwrite it.
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

**create()**

Creates a new manifest from the directory by calling `furtive.hasher.HashDirectory()` and placing the return dictionary in to *Manifest.manifest*.

**is\_empty()**

Determines if the manifest within memory is empty.

This simply checks to see if the manifest is None.

**Returns** True if manifest is empty, False otherwise.

**Return type** bool

**load()**

Load a manifest from the manifest file.

This method will open the manifest YAML file and load it in to the *manifest* object variable.

**save()**

Save the manifest to the manifest file.

Open a YAML file and dump the contents of the manifest to it.

**Requirements**

- Python 2.7, 3.4, or 3.5

**Getting Started** To install furtive, run `pip install furtive`.

**CLI Usage** See the [CLI Reference](#) for more information about available command line arguments.

**Use Case Example** Suppose you have a million digital photos in a directory called `my-photos` that you have taken over the years. You would like to know if the files begin to decay due to hardware failure or something else. Alternatively, you may wish to have reassurance that your photos have not become corrupted while being stored in a cloud backup solution such as S3 or Glacier.

To record the current state of the files, run `furtive --basedir my-photos create`

This command creates the file `.manifest.yaml` in the `my-photos/` directory. The location and name of this file can be changed by using the `--manifest` argument.

At this point, you can be sure that you will know if a file has changed. To check the files on the file system to the recorded state in the manifest, run `furtive --basedir my-photos check`. The application will output a list of files which have been added, removed, or changed. This output is YAML format so it should be easy to parse. Additionally, furtive will exit with 1 indicating the check failed. This command can be placed in a cron job and setup to send a notification if a file has changed.

**Actions** There are a few actions which can be performed by furtive.

- **create** - create a new manifest from the files in the directory specified by the `--basedir` argument.
- **compare** - compare the current state of the files on the file system with the recorded state in the manifest file. A YAML based report will be created detailing which files have changed and which files have been added or removed. Status code is 0 if the comparison was successful.
- **check** - check the integrity of files listed in the manifest. Same as `compare` but exits with status code 1 if there are changes to the files included in the manifest. That is, if any file hash changes or if files are added or removed, the application will exit with a status code of 1 to indicate there are changes. This action can be useful for scripting. For example, to run a nightly cron check of a manifest. A YAML based report will be generated as well.

**Tests** This application comes with tests. To run them, ensure you have `tox` installed (`pip install tox`). Then you can run `tox` to run the tests.

To build the docs, run `tox -e docs`. The HTML docs will be generated in the `.tox/docs/tmp/html/` directory.

**Faster YAML** By default, furtive will install and use the full Python implementation of the YAML parser which is very slow. In a testing environment, the Python implementation of the YAML loader took 1 minute to parse a 187,000 line furtive manifest file. By contrast, when the [LibYaml](#) parser was used, the loader took only 5 seconds to parse the same file.

To install the faster parser, perform the following steps:

1. Follow the [instructions from the LibYaml website](#) to download and install the latest release of libyaml.
2. Reinstall the PyYAML package by downloading the latest tar from the [PyYAML website](#) and running `python setup.py --with-libyaml install`

**CLI Reference** Command Line Interface (or Tool) reference.

Manage a Furtive manifest

```
usage: furtive [-h] [--basedir BASEDIR] [--manifest MANIFEST_PATH]
              [--log-level {debug,info,warn,error,critical}]
              [--exclude PATTERN] [--quiet] [--report-output FILE_NAME]
              [--version]
              {create,compare,check}
```

**Positional arguments:**

action	
	Which action to perform: <code>compare</code> - compare the current state of the files on the file system with the recorded state in the manifest file. Status code is 0 if the comparison was successful. <code>check</code> - check the integrity of files listed in the manifest. Same as <code>compare</code> but exits with status code 1 if there are changes to the files included in the manifest. That is, if any file hash changes or if files are added or removed, the application will exit with a status code of 1 to indicate there are changes. This action can be useful for scripting. For example, to run a nightly cron check of a manifest. <code>create</code> - create a new manifest from the files in the directory specified by the <code>--basedir</code> argument.

Possible choices: `create`, `compare`, `check`

**Options:**

<b>--basedir=.</b>	Directory containing files that will be checked. Default: .
<b>--manifest</b>	Location of the manifest file. Manifests may be located outside the directory indicated by --basedir. Must provide path and filename of the manifest file. Default: <basedir>/.manifest.yaml
<b>--log-level=info</b>	verbosity of furtive  Possible choices: debug, info, warn, error, critical
<b>--exclude=[]</b>	Patterns to exclude files and directories from manifest. Can have multiple occurrences of this argument. Excludes are not stored in the manifest so it is up to the user to provide the same arguments every run. Patterns are evaluated as UNIX shell-style wildcard characters. See the [fnmatch documentation](https://docs.python.org/2/library/fnmatch.html) for more information.  It is important to note that exclusions are not stored. Therefore, they must be specified for every run of 'furtive'. Otherwise, the files which were previously excluded will be included and will show up as files added to the manifest.
<b>--quiet=False</b>	Only print out critical error messages. Do not print a report at the end of a compare run. Using this argument will override the log-level and set it to "critical". Only exceptions will be printed to terminal. The return code will be the only way to know if a Manifest has changed. This is useful for scripting such as a cron based manifest checks. Useful with the check command.
<b>--report-output=-</b>	File to print the diff report to. - for stdout. This can be consumed by other scripts to determine exactly what has changed within the manifest. Default: -
<b>--version</b>	show program's version number and exit

## API Reference

- *API Reference*
  - *Furtive Class*
  - *Sub-Modules*
    - \* *Hasher*
    - \* *Manifest*

This document is for developers of furtive, it contains the API functions

## Furtive Class Furtive - File Integrity Verification System

**class** furtive.**Furtive**(*base\_dir*, *manifest\_path*, *exclude=None*)  
Bases: object

Furtive is an application which stores file state and allows users to verify the state in the future. Example use cases include file archives and file transport.

If the manifest file exists, it will be automatically loaded. Calling create() will overwrite the existing manifest in memory as well as the file.

### Parameters

- **base\_dir** (*str*) – Base directory to use for the manifest. Can be a full or relative path.



- **manifest\_path** (*str*) – Path to the manifest file. Can be a full or relative path.
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

**compare()**

Compare the hashes in the database with the current hashes of files on the file system.

**Returns** Dictionary of added, deleted, and changed files.

**Return type** dict

**create()**

Create and save a new manifest.

The contents of the new Manifest() will be saved to *manifest\_path*.

**Returns** None

**Sub-Modules**

**Hasher** Manages the hashing of files

**class** furtive.hasher.HashDirectory (*directory*, *exclude=None*)

Bases: object

Object to manage hashing files in a directory.

This object is responsible for walking the directory tree and adding each file to a list. Once the directory walk has completed, each file path is passed to hash\_task(). After each file has been hashed, this object will then create a Python dictionary of files with their associated hash.

**Parameters**

- **directory** (*str*) – Path to directory containing files
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

**Returns** Dictionary of file:hash

**Return type** dict

**excluded(file\_path)**

Should the file be excluded from the manifest?

Determines if a file should be excluded based on UNIX style pattern matching. Think \*, ?, and [] sequences.

For matchers, see <https://docs.python.org/2/library/fnmatch.html>

**Parameters** **file\_path** (*str*) – path of the file to match against.

**Returns** True or False indicating if the file should be excluded from the list of files contained within the manifest.

**Return type** bool

**hash\_files()**

Orchestrates the discovery and hashing of files.

Note: This method only supports the md5 hashing algorithm

furtive.hasher.hash\_task (*file\_path*, *hash\_algorithm='md5'*)

Responsible for hashing a file.

This function reads in the file `file_path` in small chunks the size of the hash algorithm's block size in order to avoid running out of memory. This means that this function should be able to read any file regardless of the size.

**Parameters**

- **file\_path** (*str*) – path of file to hash
- **hash\_algorithm** (*str*) – the hashing algorithm to use. All options available in *hashlib* should work. See: <https://docs.python.org/2/library/hashlib.html>

**Returns** hash of file

**Return type** dict

`furtive.hasher.initializer(terminating_)`

Method to make terminating a global variable so that it is inherited by child processes.

**Manifest** Manifest of files and their hashes

**class** `furtive.manifest.Manifest(directory, manifest_file, exclude=None)`

Bases: object

Manifest of files and the associated hashes.

**Parameters**

- **directory** – directory which will serve as the root for the manifest. All files under the directory will be hashed and added to or compared with the manifest.
- **type** – str
- **manifest\_file** – file location of the manifest file. This is the path which will be used for the `create()` and `compare()` methods. If the file exists, the `create()` method will overwrite it.
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

**create()**

Creates a new manifest from the directory by calling `furtive.hasher.HashDirectory()` and placing the return dictionary in to *Manifest.manifest*.

**is\_empty()**

Determines if the manifest within memory is empty.

This simply checks to see if the manifest is None.

**Returns** True if manifest is empty, False otherwise.

**Return type** bool

**load()**

Load a manifest from the manifest file.

This method will open the manifest YAML file and load it in to the *manifest* object variable.

**save()**

Save the manifest to the manifest file.

Open a YAML file and dump the contents of the manifest to it.

**Requirements**

- Python 2.7, 3.4, or 3.5

**Getting Started** To install furtive, run `pip install furtive`.

**CLI Usage** See the [CLI Reference](#) for more information about available command line arguments.

**Use Case Example** Suppose you have a million digital photos in a directory called `my-photos` that you have taken over the years. You would like to know if the files begin to decay due to hardware failure or something else. Alternatively, you may wish to have reassurance that your photos have not become corrupted while being stored in a cloud backup solution such as S3 or Glacier.

To record the current state of the files, run `furtive --basedir my-photos create`

This command creates the file `.manifest.yaml` in the `my-photos/` directory. The location and name of this file can be changed by using the `--manifest` argument.

At this point, you can be sure that you will know if a file has changed. To check the files on the file system to the recorded state in the manifest, run `furtive --basedir my-photos check`. The application will output a list of files which have been added, removed, or changed. This output is YAML format so it should be easy to parse. Additionally, furtive will exit with 1 indicating the check failed. This command can be placed in a cron job and setup to send a notification if a file has changed.

**Actions** There are a few actions which can be performed by furtive.

- **create** - create a new manifest from the files in the directory specified by the `--basedir` argument.
- **compare** - compare the current state of the files on the file system with the recorded state in the manifest file. A YAML based report will be created detailing which files have changed and which files have been added or removed. Status code is 0 if the comparison was successful.
- **check** - check the integrity of files listed in the manifest. Same as `compare` but exits with status code 1 if there are changes to the files included in the manifest. That is, if any file hash changes or if files are added or removed, the application will exit with a status code of 1 to indicate there are changes. This action can be useful for scripting. For example, to run a nightly cron check of a manifest. A YAML based report will be generated as well.

**Tests** This application comes with tests. To run them, ensure you have `tox` installed (`pip install tox`). Then you can run `tox` to run the tests.

To build the docs, run `tox -e docs`. The HTML docs will be generated in the `.tox/docs/tmp/html/` directory.

**Faster YAML** By default, furtive will install and use the full Python implementation of the YAML parser which is very slow. In a testing environment, the Python implementation of the YAML loader took 1 minute to parse a 187,000 line furtive manifest file. By contrast, when the [LibYaml](#) parser was used, the loader took only 5 seconds to parse the same file.

To install the faster parser, perform the following steps:

1. Follow the [instructions from the LibYaml website](#) to download and install the latest release of libyaml.
2. Reinstall the PyYAML package by downloading the latest tar from the [PyYAML website](#) and running `python setup.py --with-libyaml install`

**CLI Reference** Command Line Interface (or Tool) reference.

## Manage a Furtive manifest

```
usage: furtive [-h] [--basedir BASEDIR] [--manifest MANIFEST_PATH]
              [--log-level {debug,info,warn,error,critical}]
              [--exclude PATTERN] [--quiet] [--report-output FILE_NAME]
              [--version]
              {create,compare,check}
```

**Positional arguments:**

<b>action</b>	Which action to perform: compare - compare the current state of the files on the file system with the recorded state in the manifest file. Status code is 0 if the comparison was successful. check - check the integrity of files listed in the manifest. Same as compare but exits with status code 1 if there are changes to the files included in the manifest. That is, if any file hash changes or if files are added or removed, the application will exit with a status code of 1 to indicate there are changes. This action can be useful for scripting. For example, to run a nightly cron check of a manifest. create - create a new manifest from the files in the directory specified by the --basedir argument.
---------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Possible choices: create, compare, check

**Options:**

<b>--basedir=.</b>	Directory containing files that will be checked. Default: .
<b>--manifest</b>	Location of the manifest file. Manifests may be located outside the directory indicated by --basedir. Must provide path and filename of the manifest file. Default: <basedir>/manifest.yaml
<b>--log-level=info</b>	verbosity of furtive  Possible choices: debug, info, warn, error, critical
<b>--exclude=[]</b>	Patterns to exclude files and directories from manifest. Can have multiple occurrences of this argument. Excludes are not stored in the manifest so it is up to the user to provide the same arguments every run. Patterns are evaluated as UNIX shell-style wildcard characters. See the [fnmatch documentation](https://docs.python.org/2/library/fnmatch.html) for more information.  It is important to note that exclusions are not stored. Therefore, they must be specified for every run of 'furtive'. Otherwise, the files which were previously excluded will be included and will show up as files added to the manifest.
<b>--quiet=False</b>	Only print out critical error messages. Do not print a report at the end of a compare run. Using this argument will override the log-level and set it to "critical". Only exceptions will be printed to terminal. The return code will be the only way to know if a Manifest has changed. This is useful for scripting such as a cron based manifest checks. Useful with the check command.
<b>--report-output=-</b>	File to print the diff report to. - for stdout. This can be consumed by other scripts to determine exactly what has changed within the manifest. Default: -
<b>--version</b>	show program's version number and exit

## API Reference

- *API Reference*
  - *Furtive Class*
  - *Sub-Modules*
    - \* *Hasher*
    - \* *Manifest*

This document is for developers of furtive, it contains the API functions

### Furtive Class Furtive - File Integrity Verification System

**class** `furtive.Furtive` (*base\_dir*, *manifest\_path*, *exclude=None*)

Bases: `object`

Furtive is an application which stores file state and allows users to verify the state in the future. Example use cases include file archives and file transport.

If the manifest file exists, it will be automatically loaded. Calling `create()` will overwrite the existing manifest in memory as well as the file.

#### Parameters

- **base\_dir** (*str*) – Base directory to use for the manifest. Can be a full or relative path.
- **manifest\_path** (*str*) – Path to the manifest file. Can be a full or relative path.
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

**compare** ()

Compare the hashes in the database with the current hashes of files on the file system.

**Returns** Dictionary of added, deleted, and changed files.

**Return type** `dict`

**create** ()

Create and save a new manifest.

The contents of the new Manifest() will be saved to *manifest\_path*.

**Returns** `None`

## Sub-Modules

### Hasher Manages the hashing of files

**class** `furtive.hasher.HashDirectory` (*directory*, *exclude=None*)

Bases: `object`

Object to manage hashing files in a directory.

This object is responsible for walking the directory tree and adding each file to a list. Once the directory walk has completed, each file path is passed to `hash_task()`. After each file has been hashed, this object will then create a Python dictionary of files with their associated hash.

#### Parameters

- **directory** (*str*) – Path to directory containing files

- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

**Returns** Dictionary of file:hash

**Return type** dict

**excluded** (*file\_path*)

Should the file be excluded from the manifest?

Determines if a file should be excluded based on UNIX style pattern matching. Think `*`, `?`, and `[]` sequences.

For matchers, see <https://docs.python.org/2/library/fnmatch.html>

**Parameters** **file\_path** (*str*) – path of the file to match against.

**Returns** True or False indicating if the file should be excluded from the list of files contained within the manifest.

**Return type** bool

**hash\_files** ()

Orchestrates the discovery and hashing of files.

Note: This method only supports the md5 hashing algorithm

`furtive.hasher.hash_task` (*file\_path*, *hash\_algorithm*='md5')

Responsible for hashing a file.

This function reads in the file *file\_path* in small chunks the size of the hash algorithm's block size in order to avoid running out of memory. This means that this function should be able to read any file irregardless of the size.

**Parameters**

- **file\_path** (*str*) – path of file to hash
- **hash\_algorithm** (*str*) – the hashing algorithm to use. All options available in *hashlib* should work. See: <https://docs.python.org/2/library/hashlib.html>

**Returns** hash of file

**Return type** dict

`furtive.hasher.initializer` (*terminating\_*)

Method to make terminating a global variable so that it is inherited by child processes.

**Manifest** Manifest of files and their hashes

**class** `furtive.manifest.Manifest` (*directory*, *manifest\_file*, *exclude*=None)

Bases: object

Manifest of files and the associated hashes.

**Parameters**

- **directory** – directory which will serve as the root for the manifest. All files under the directory will be hashed and added to or compared with the manifest.
- **type** – str
- **manifest\_file** – file location of the manifest file. This is the path which will be used for the `create()` and `compare()` methods. If the file exists, the `create()` method will overwrite it.
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

**create()**

Creates a new manifest from the directory by calling `furtive.hasher.HashDirectory()` and placing the return dictionary in to *Manifest.manifest*.

**is\_empty()**

Determines if the manifest within memory is empty.

This simply checks to see if the manifest is None.

**Returns** True if manifest is empty, False otherwise.

**Return type** bool

**load()**

Load a manifest from the manifest file.

This method will open the manifest YAML file and load it in to the *manifest* object variable.

**save()**

Save the manifest to the manifest file.

Open a YAML file and dump the contents of the manifest to it.

**Requirements**

- Python 2.7, 3.4, or 3.5

**Getting Started** To install furtive, run `pip install furtive`.

**CLI Usage** See the [CLI Reference](#) for more information about available command line arguments.

**Use Case Example** Suppose you have a million digital photos in a directory called `my-photos` that you have taken over the years. You would like to know if the files begin to decay due to hardware failure or something else. Alternatively, you may wish to have reassurance that your photos have not become corrupted while being stored in a cloud backup solution such as S3 or Glacier.

To record the current state of the files, run `furtive --basedir my-photos create`

This command creates the file `.manifest.yaml` in the `my-photos/` directory. The location and name of this file can be changed by using the `--manifest` argument.

At this point, you can be sure that you will know if a file has changed. To check the files on the file system to the recorded state in the manifest, run `furtive --basedir my-photos check`. The application will output a list of files which have been added, removed, or changed. This output is YAML format so it should be easy to parse. Additionally, furtive will exit with 1 indicating the check failed. This command can be placed in a cron job and setup to send a notification if a file has changed.

**Actions** There are a few actions which can be performed by furtive.

- **create** - create a new manifest from the files in the directory specified by the `--basedir` argument.
- **compare** - compare the current state of the files on the file system with the recorded state in the manifest file. A YAML based report will be created detailing which files have changed and which files have been added or removed. Status code is 0 if the comparison was successful.
- **check** - check the integrity of files listed in the manifest. Same as `compare` but exits with status code 1 if there are changes to the files included in the manifest. That is, if any file hash changes or if files are added or removed, the application will exit with a status code of 1 to indicate there are changes. This action can be useful

for scripting. For example, to run a nightly cron check of a manifest. A YAML based report will be generated as well.

**Tests** This application comes with tests. To run them, ensure you have `tox` installed (`pip install tox`). Then you can run `tox` to run the tests.

To build the docs, run `tox -e docs`. The HTML docs will be generated in the `.tox/docs/tmp/html/` directory.

**Faster YAML** By default, furtive will install and use the full Python implementation of the YAML parser which is very slow. In a testing environment, the Python implementation of the YAML loader took 1 minute to parse a 187,000 line furtive manifest file. By contrast, when the [LibYaml](#) parser was used, the loader took only 5 seconds to parse the same file.

To install the faster parser, perform the following steps:

1. Follow the [instructions from the LibYaml website](#) to download and install the latest release of libyaml.
2. Reinstall the PyYAML package by downloading the latest tar from the [PyYAML website](#) and running `python setup.py --with-libyaml install`

**CLI Reference** Command Line Interface (or Tool) reference.

Manage a Furtive manifest

```
usage: furtive [-h] [--basedir BASEDIR] [--manifest MANIFEST_PATH]
              [--log-level {debug,info,warn,error,critical}]
              [--exclude PATTERN] [--quiet] [--report-output FILE_NAME]
              [--version]
              {create,compare,check}
```

**Positional arguments:**

<b>action</b>	Which action to perform: <code>compare</code> - compare the current state of the files on the file system with the recorded state in the manifest file. Status code is 0 if the comparison was successful. <code>check</code> - check the integrity of files listed in the manifest. Same as <code>compare</code> but exits with status code 1 if there are changes to the files included in the manifest. That is, if any file hash changes or if files are added or removed, the application will exit with a status code of 1 to indicate there are changes. This action can be useful for scripting. For example, to run a nightly cron check of a manifest. <code>create</code> - create a new manifest from the files in the directory specified by the <code>--basedir</code> argument.
---------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Possible choices: `create`, `compare`, `check`

**Options:**

<b>--basedir=.</b>	Directory containing files that will be checked. Default: <code>.</code>
<b>--manifest</b>	Location of the manifest file. Manifests may be located outside the directory indicated by <code>--basedir</code> . Must provide path and filename of the manifest file. Default: <code>&lt;basedir&gt;/manifest.yaml</code>
<b>--log-level=info</b>	verbosity of furtive

Possible choices: `debug`, `info`, `warn`, `error`, `critical`



<b>--exclude=[]</b>	Patterns to exclude files and directories from manifest. Can have multiple occurrences of this argument. Excludes are not stored in the manifest so it is up to the user to provide the same arguments every run. Patterns are evaluated as UNIX shell-style wildcard characters. See the [fnmatch documentation](https://docs.python.org/2/library/fnmatch.html) for more information.  It is important to note that exclusions are not stored. Therefore, they must be specified for every run of 'furtive'. Otherwise, the files which were previously excluded will be included and will show up as files added to the manifest.
<b>--quiet=False</b>	Only print out critical error messages. Do not print a report at the end of a compare run. Using this argument will override the log-level and set it to "critical". Only exceptions will be printed to terminal. The return code will be the only way to know if a Manifest has changed. This is useful for scripting such as a cron based manifest checks. Useful with the check command.
<b>--report-output=-</b>	File to print the diff report to. - for stdout. This can be consumed by other scripts to determine exactly what has changed within the manifest. Default: -
<b>--version</b>	show program's version number and exit

## API Reference

- *API Reference*
  - *Furtive Class*
  - *Sub-Modules*
    - \* *Hasher*
    - \* *Manifest*

This document is for developers of furtive, it contains the API functions

## Furtive Class Furtive - File Integrity Verification System

**class** `furtive.Furtive` (*base\_dir*, *manifest\_path*, *exclude=None*)  
Bases: `object`

Furtive is an application which stores file state and allows users to verify the state in the future. Example uses include file archives and file transport.

If the manifest file exists, it will be automatically loaded. Calling `create()` will overwrite the existing manifest in memory as well as the file.

### Parameters

- **base\_dir** (*str*) – Base directory to use for the manifest. Can be a full or relative path.
- **manifest\_path** (*str*) – Path to the manifest file. Can be a full or relative path.
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

### `compare()`

Compare the hashes in the database with the current hashes of files on the file system.

**Returns** Dictionary of added, deleted, and changed files.

**Return type** dict

**create()**

Create and save a new manifest.

The contents of the new Manifest() will be saved to *manifest\_path*.

**Returns** None

**Sub-Modules**

**Hasher** Manages the hashing of files

**class** furtive.hasher.**HashDirectory**(*directory*, *exclude=None*)

Bases: object

Object to manage hashing files in a directory.

This object is responsible for walking the directory tree and adding each file to a list. Once the directory walk has completed, each file path is passed to `hash_task()`. After each file has been hashed, this object will then create a Python dictionary of files with their associated hash.

**Parameters**

- **directory** (*str*) – Path to directory containing files
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

**Returns** Dictionary of file:hash

**Return type** dict

**excluded**(*file\_path*)

Should the file be excluded from the manifest?

Determines if a file should be excluded based on UNIX style pattern matching. Think `*`, `?`, and `[]` sequences.

For matchers, see <https://docs.python.org/2/library/fnmatch.html>

**Parameters** **file\_path** (*str*) – path of the file to match against.

**Returns** True or False indicating if the file should be excluded from the list of files contained within the manifest.

**Return type** bool

**hash\_files()**

Orchestrates the discovery and hashing of files.

Note: This method only supports the md5 hashing algorithm

**furtive.hasher.hash\_task**(*file\_path*, *hash\_algorithm='md5'*)

Responsible for hashing a file.

This function reads in the file *file\_path* in small chunks the size of the hash algorithm's block size in order to avoid running out of memory. This means that this function should be able to read any file regardless of the size.

**Parameters**

- **file\_path** (*str*) – path of file to hash
- **hash\_algorithm** (*str*) – the hashing algorithm to use. All options available in *hashlib* should work. See: <https://docs.python.org/2/library/hashlib.html>

**Returns** hash of file

**Return type** dict

`furtive.hasher.initializer(terminating_)`

Method to make terminating a global variable so that it is inherited by child processes.

**Manifest** Manifest of files and their hashes

**class** `furtive.manifest.Manifest(directory, manifest_file, exclude=None)`

Bases: object

Manifest of files and the associated hashes.

#### Parameters

- **directory** – directory which will serve as the root for the manifest. All files under the directory will be hashed and added to or compared with the manifest.
- **type** – str
- **manifest\_file** – file location of the manifest file. This is the path which will be used for the `create()` and `compare()` methods. If the file exists, the `create()` method will overwrite it.
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

**create()**

Creates a new manifest from the directory by calling `furtive.hasher.HashDirectory()` and placing the return dictionary in to *Manifest.manifest*.

**is\_empty()**

Determines if the manifest within memory is empty.

This simply checks to see if the manifest is None.

**Returns** True if manifest is empty, False otherwise.

**Return type** bool

**load()**

Load a manifest from the manifest file.

This method will open the manifest YAML file and load it in to the *manifest* object variable.

**save()**

Save the manifest to the manifest file.

Open a YAML file and dump the contents of the manifest to it.

#### Requirements

- Python 2.7, 3.4, or 3.5

**Getting Started** To install furtive, run `pip install furtive`.

**CLI Usage** See the [CLI Reference](#) for more information about available command line arguments.

**Use Case Example** Suppose you have a million digital photos in a directory called `my-photos` that you have taken over the years. You would like to know if the files begin to decay due to hardware failure or something else. Alternatively, you may wish to have reassurance that your photos have not become corrupted while being stored in a cloud backup solution such as S3 or Glacier.

To record the current state of the files, run `furtive --basedir my-photos create`

This command creates the file `.manifest.yaml` in the `my-photos/` directory. The location and name of this file can be changed by using the `--manifest` argument.

At this point, you can be sure that you will know if a file has changed. To check the files on the file system to the recorded state in the manifest, run `furtive --basedir my-photos check`. The application will output a list of files which have been added, removed, or changed. This output is YAML format so it should be easy to parse. Additionally, `furtive` will exit with 1 indicating the check failed. This command can be placed in a cron job and setup to send a notification if a file has changed.

**Actions** There are a few actions which can be performed by `furtive`.

- **create** - create a new manifest from the files in the directory specified by the `--basedir` argument.
- **compare** - compare the current state of the files on the file system with the recorded state in the manifest file. A YAML based report will be created detailing which files have changed and which files have been added or removed. Status code is 0 if the comparison was successful.
- **check** - check the integrity of files listed in the manifest. Same as `compare` but exits with status code 1 if there are changes to the files included in the manifest. That is, if any file hash changes or if files are added or removed, the application will exit with a status code of 1 to indicate there are changes. This action can be useful for scripting. For example, to run a nightly cron check of a manifest. A YAML based report will be generated as well.

**Tests** This application comes with tests. To run them, ensure you have `tox` installed (`pip install tox`). Then you can run `tox` to run the tests.

To build the docs, run `tox -e docs`. The HTML docs will be generated in the `.tox/docs/tmp/html/` directory.

**Faster YAML** By default, `furtive` will install and use the full Python implementation of the YAML parser which is very slow. In a testing environment, the Python implementation of the YAML loader took 1 minute to parse a 187,000 line `furtive` manifest file. By contrast, when the [LibYaml](#) parser was used, the loader took only 5 seconds to parse the same file.

To install the faster parser, perform the following steps:

1. Follow the [instructions from the LibYaml website](#) to download and install the latest release of `libyaml`.
2. Reinstall the `PyYAML` package by downloading the latest tar from the [PyYAML website](#) and running `python setup.py --with-libyaml install`

**CLI Reference** Command Line Interface (or Tool) reference.

Manage a Furtive manifest

```
usage: furtive [-h] [--basedir BASEDIR] [--manifest MANIFEST_PATH]
              [--log-level {debug,info,warn,error,critical}]
              [--exclude PATTERN] [--quiet] [--report-output FILE_NAME]
              [--version]
              {create,compare,check}
```

**Positional arguments:**

**action** Which action to perform: compare - compare the current state of the files on the file system with the recorded state in the manifest file. Status code is 0 if the comparison was successful. check - check the integrity of files listed in the manifest. Same as compare but exits with status code 1 if there are changes to the files included in the manifest. That is, if any file hash changes or if files are added or removed, the application will exit with a status code of 1 to indicate there are changes. This action can be useful for scripting. For example, to run a nightly cron check of a manifest. create - create a new manifest from the files in the directory specified by the --basedir argument.

Possible choices: create, compare, check

**Options:**

**--basedir=.** Directory containing files that will be checked. Default: .

**--manifest** Location of the manifest file. Manifests may be located outside the directory indicated by --basedir. Must provide path and filename of the manifest file. Default: <basedir>/manifest.yaml

**--log-level=info** verbosity of furtive

Possible choices: debug, info, warn, error, critical

**--exclude=[]** Patterns to exclude files and directories from manifest. Can have multiple occurrences of this argument. Excludes are not stored in the manifest so it is up to the user to provide the same arguments every run. Patterns are evaluated as UNIX shell-style wildcard characters. See the [fnmatch documentation](https://docs.python.org/2/library/fnmatch.html) for more information.

It is important to note that exclusions are not stored. Therefore, they must be specified for every run of 'furtive'. Otherwise, the files which were previously excluded will be included and will show up as files added to the manifest.

**--quiet=False** Only print out critical error messages. Do not print a report at the end of a compare run. Using this argument will override the log-level and set it to "critical". Only exceptions will be printed to terminal. The return code will be the only way to know if a Manifest has changed. This is useful for scripting such as a cron based manifest checks. Useful with the check command.

**--report-output=-** File to print the diff report to. - for stdout. This can be consumed by other scripts to determine exactly what has changed within the manifest. Default: -

**--version** show program's version number and exit

**API Reference**

- *API Reference*
  - *Furtive Class*
  - *Sub-Modules*
    - \* *Hasher*
    - \* *Manifest*

This document is for developers of furtive, it contains the API functions

### Furtive Class Furtive - File Integrity Verification System

**class** furtive.**Furtive** (*base\_dir*, *manifest\_path*, *exclude=None*)

Bases: object

Furtive is an application which stores file state and allows users to verify the state in the future. Example use cases include file archives and file transport.

If the manifest file exists, it will be automatically loaded. Calling create() will overwrite the existing manifest in memory as well as the file.

#### Parameters

- **base\_dir** (*str*) – Base directory to use for the manifest. Can be a full or relative path.
- **manifest\_path** (*str*) – Path to the manifest file. Can be a full or relative path.
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

**compare** ()

Compare the hashes in the database with the current hashes of files on the file system.

**Returns** Dictionary of added, deleted, and changed files.

**Return type** dict

**create** ()

Create and save a new manifest.

The contents of the new Manifest() will be saved to *manifest\_path*.

**Returns** None

### Sub-Modules

**Hasher** Manages the hashing of files

**class** furtive.hasher.**HashDirectory** (*directory*, *exclude=None*)

Bases: object

Object to manage hashing files in a directory.

This object is responsible for walking the directory tree and adding each file to a list. Once the directory walk has completed, each file path is passed to hash\_task(). After each file has been hashed, this object will then create a Python dictionary of files with their associated hash.

#### Parameters

- **directory** (*str*) – Path to directory containing files
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

**Returns** Dictionary of file:hash

**Return type** dict

**excluded** (*file\_path*)

Should the file be excluded from the manifest?

Determines if a file should be excluded based on UNIX style pattern matching. Think \*, ?, and [] sequences.

For matchers, see <https://docs.python.org/2/library/fnmatch.html>

**Parameters** `file_path` (*str*) – path of the file to match against.

**Returns** True or False indicating if the file should be excluded from the list of files contained within the manifest.

**Return type** bool

**hash\_files** ()

Orchestrates the discovery and hashing of files.

Note: This method only supports the md5 hashing algorithm

`furtive.hasher.hash_task` (*file\_path*, *hash\_algorithm*='md5')

Responsible for hashing a file.

This function reads in the file `file_path` in small chunks the size of the hash algorithm's block size in order to avoid running out of memory. This means that this function should be able to read any file regardless of the size.

**Parameters**

- **file\_path** (*str*) – path of file to hash
- **hash\_algorithm** (*str*) – the hashing algorithm to use. All options available in *hashlib.algorithms* should work. See: <https://docs.python.org/2/library/hashlib.html>

**Returns** hash of file

**Return type** dict

`furtive.hasher.initializer` (*terminating\_*)

Method to make terminating a global variable so that it is inherited by child processes.

**Manifest** Manifest of files and their hashes

**class** `furtive.manifest.Manifest` (*directory*, *manifest\_file*, *exclude*=None)

Bases: object

Manifest of files and the associated hashes.

**Parameters**

- **directory** – directory which will serve as the root for the manifest. All files under the directory will be hashed and added to or compared with the manifest.
- **type** – str
- **manifest\_file** – file location of the manifest file. This is the path which will be used for the `create()` and `compare()` methods. If the file exists, the `create()` method will overwrite it.
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

**create** ()

Creates a new manifest from the directory by calling `furtive.hasher.HashDirectory()` and placing the return dictionary in to *Manifest.manifest*.

**is\_empty** ()

Determines if the manifest within memory is empty.

This simply checks to see if the manifest is None.

**Returns** True if manifest is empty, False otherwise.

**Return type** bool

**load()**

Load a manifest from the manifest file.

This method will open the manifest YAML file and load it in to the *manifest* object variable.

**save()**

Save the manifest to the manifest file.

Open a YAML file and dump the contents of the manifest to it.

### Requirements

- Python 2.7, 3.4, or 3.5

**Getting Started** To install furtive, run `pip install furtive`.

**CLI Usage** See the [CLI Reference](#) for more information about available command line arguments.

**Use Case Example** Suppose you have a million digital photos in a directory called `my-photos` that you have taken over the years. You would like to know if the files begin to decay due to hardware failure or something else. Alternatively, you may wish to have reassurance that your photos have not become corrupted while being stored in a cloud backup solution such as S3 or Glacier.

To record the current state of the files, run `furtive --basedir my-photos create`

This command creates the file `.manifest.yaml` in the `my-photos/` directory. The location and name of this file can be changed by using the `--manifest` argument.

At this point, you can be sure that you will know if a file has changed. To check the files on the file system to the recorded state in the manifest, run `furtive --basedir my-photos check`. The application will output a list of files which have been added, removed, or changed. This output is YAML format so it should be easy to parse. Additionally, furtive will exit with 1 indicating the check failed. This command can be placed in a cron job and setup to send a notification if a file has changed.

**Actions** There are a few actions which can be performed by furtive.

- **create** - create a new manifest from the files in the directory specified by the `--basedir` argument.
- **compare** - compare the current state of the files on the file system with the recorded state in the manifest file. A YAML based report will be created detailing which files have changed and which files have been added or removed. Status code is 0 if the comparison was successful.
- **check** - check the integrity of files listed in the manifest. Same as `compare` but exits with status code 1 if there are changes to the files included in the manifest. That is, if any file hash changes or if files are added or removed, the application will exit with a status code of 1 to indicate there are changes. This action can be useful for scripting. For example, to run a nightly cron check of a manifest. A YAML based report will be generated as well.

**Tests** This application comes with tests. To run them, ensure you have `tox` installed (`pip install tox`). Then you can run `tox` to run the tests.

To build the docs, run `tox -e docs`. The HTML docs will be generated in the `.tox/docs/tmp/html/` directory.



**Faster YAML** By default, furtive will install and use the full Python implementation of the YAML parser which is very slow. In a testing environment, the Python implementation of the YAML loader took 1 minute to parse a 187,000 line furtive manifest file. By contrast, when the [LibYaml](#) parser was used, the loader took only 5 seconds to parse the same file.

To install the faster parser, perform the following steps:

1. Follow the [instructions from the LibYaml website](#) to download and install the latest release of libyaml.
2. Reinstall the PyYAML package by downloading the latest tar from the [PyYAML website](#) and running `python setup.py --with-libyaml install`

**CLI Reference** Command Line Interface (or Tool) reference.

Manage a Furtive manifest

```
usage: furtive [-h] [--basedir BASEDIR] [--manifest MANIFEST_PATH]
              [--log-level {debug,info,warn,error,critical}]
              [--exclude PATTERN] [--quiet] [--report-output FILE_NAME]
              [--version]
              {create,compare,check}
```

**Positional arguments:**

<b>action</b>	Which action to perform: compare - compare the current state of the files on the file system with the recorded state in the manifest file. Status code is 0 if the comparison was successful. check - check the integrity of files listed in the manifest. Same as compare but exits with status code 1 if there are changes to the files included in the manifest. That is, if any file hash changes or if files are added or removed, the application will exit with a status code of 1 to indicate there are changes. This action can be useful for scripting. For example, to run a nightly cron check of a manifest. create - create a new manifest from the files in the directory specified by the --basedir argument.
---------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Possible choices: create, compare, check

**Options:**

<b>--basedir=.</b>	Directory containing files that will be checked. Default: .
<b>--manifest</b>	Location of the manifest file. Manifests may be located outside the directory indicated by --basedir. Must provide path and filename of the manifest file. Default: <basedir>/manifest.yaml
<b>--log-level=info</b>	verbosity of furtive Possible choices: debug, info, warn, error, critical
<b>--exclude=[]</b>	Patterns to exclude files and directories from manifest. Can have multiple occurrences of this argument. Excludes are not stored in the manifest so it is up to the user to provide the same arguments every run. Patterns are evaluated as UNIX shell-style wildcard characters. See the <a href="https://docs.python.org/2/library/fnmatch.html">[fnmatch documentation](https://docs.python.org/2/library/fnmatch.html)</a> for more information.

It is important to note that exclusions are not stored. Therefore, they must be specified for every run of 'furtive'. Otherwise, the files which were previously excluded will be included and will show up as files added to the manifest.

<b>--quiet=False</b>	Only print out critical error messages. Do not print a report at the end of a compare run. Using this argument will override the log-level and set it to “critical”. Only exceptions will be printed to terminal. The return code will be the only way to know if a Manifest has changed. This is useful for scripting such as a cron based manifest checks. Useful with the check command.
<b>--report-output=-</b>	File to print the diff report to. - for stdout. This can be consumed by other scripts to determine exactly what has changed within the manifest. Default: -
<b>--version</b>	show program’s version number and exit

### API Reference

- *API Reference*
  - *Furtive Class*
  - *Sub-Modules*
    - \* *Hasher*
    - \* *Manifest*

This document is for developers of furtive, it contains the API functions

### Furtive Class Furtive - File Integrity Verification System

**class** `furtive.Furtive` (*base\_dir*, *manifest\_path*, *exclude=None*)

Bases: `object`

Furtive is an application which stores file state and allows users to verify the state in the future. Example use cases include file archives and file transport.

If the manifest file exists, it will be automatically loaded. Calling `create()` will overwrite the existing manifest in memory as well as the file.

#### Parameters

- **base\_dir** (*str*) – Base directory to use for the manifest. Can be a full or relative path.
- **manifest\_path** (*str*) – Path to the manifest file. Can be a full or relative path.
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

#### **compare** ()

Compare the hashes in the database with the current hashes of files on the file system.

**Returns** Dictionary of added, deleted, and changed files.

**Return type** dict

#### **create** ()

Create and save a new manifest.

The contents of the new Manifest() will be saved to *manifest\_path*.

**Returns** None

### Sub-Modules

**Hasher** Manages the hashing of files

**class** `furtive.hasher.HashDirectory(directory, exclude=None)`

Bases: `object`

Object to manage hashing files in a directory.

This object is responsible for walking the directory tree and adding each file to a list. Once the directory walk has completed, each file path is passed to `hash_task()`. After each file has been hashed, this object will then create a Python dictionary of files with their associated hash.

**Parameters**

- **directory** (*str*) – Path to directory containing files
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

**Returns** Dictionary of file:hash

**Return type** `dict`

**excluded** (*file\_path*)

Should the file be excluded from the manifest?

Determines if a file should be excluded based on UNIX style pattern matching. Think `*`, `?`, and `[]` sequences.

For matchers, see <https://docs.python.org/2/library/fnmatch.html>

**Parameters** **file\_path** (*str*) – path of the file to match against.

**Returns** True or False indicating if the file should be excluded from the list of files contained within the manifest.

**Return type** `bool`

**hash\_files** ()

Orchestrates the discovery and hashing of files.

Note: This method only supports the md5 hashing algorithm

`furtive.hasher.hash_task(file_path, hash_algorithm='md5')`

Responsible for hashing a file.

This function reads in the file `file_path` in small chunks the size of the hash algorithm's block size in order to avoid running out of memory. This means that this function should be able to read any file regardless of the size.

**Parameters**

- **file\_path** (*str*) – path of file to hash
- **hash\_algorithm** (*str*) – the hashing algorithm to use. All options available in `hashlib.algorithms` should work. See: <https://docs.python.org/2/library/hashlib.html>

**Returns** hash of file

**Return type** `dict`

`furtive.hasher.initializer(terminating_)`

Method to make terminating a global variable so that it is inherited by child processes.

**Manifest** Manifest of files and their hashes

**class** `furtive.manifest.Manifest` (*directory*, *manifest\_file*, *exclude=None*)

Bases: `object`

Manifest of files and the associated hashes.

### Parameters

- **directory** – directory which will serve as the root for the manifest. All files under the directory will be hashed and added to or compared with the manifest.
- **type** – str
- **manifest\_file** – file location of the manifest file. This is the path which will be used for the `create()` and `compare()` methods. If the file exists, the `create()` method will overwrite it.
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

**create** ()

Creates a new manifest from the directory by calling `furtive.hasher.HashDirectory()` and placing the return dictionary in to *Manifest.manifest*.

**is\_empty** ()

Determines if the manifest within memory is empty.

This simply checks to see if the manifest is `None`.

**Returns** True if manifest is empty, False otherwise.

**Return type** bool

**load** ()

Load a manifest from the manifest file.

This method will open the manifest YAML file and load it in to the *manifest* object variable.

**save** ()

Save the manifest to the manifest file.

Open a YAML file and dump the contents of the manifest to it.

## Requirements

- Python 2.7, 3.4, or 3.5

**Getting Started** To install furtive, run `pip install furtive`.

**CLI Usage** See the [CLI Reference](#) for more information about available command line arguments.

**Use Case Example** Suppose you have a million digital photos in a directory called `my-photos` that you have taken over the years. You would like to know if the files begin to decay due to hardware failure or something else. Alternatively, you may wish to have reassurance that your photos have not become corrupted while being stored in a cloud backup solution such as S3 or Glacier.

To record the current state of the files, run `furtive --basedir my-photos create`

This command creates the file `.manifest.yaml` in the `my-photos/` directory. The location and name of this file can be changed by using the `--manifest` argument.

At this point, you can be sure that you will know if a file has changed. To check the files on the file system to the recorded state in the manifest, run `furtive --basedir my-photos check`. The application will output a list of files which have been added, removed, or changed. This output is YAML format so it should be easy to parse. Additionally, furtive will exit with 1 indicating the check failed. This command can be placed in a cron job and setup to send a notification if a file has changed.

**Actions** There are a few actions which can be performed by furtive.

- **create** - create a new manifest from the files in the directory specified by the `--basedir` argument.
- **compare** - compare the current state of the files on the file system with the recorded state in the manifest file. A YAML based report will be created detailing which files have changed and which files have been added or removed. Status code is 0 if the comparison was successful.
- **check** - check the integrity of files listed in the manifest. Same as `compare` but exits with status code 1 if there are changes to the files included in the manifest. That is, if any file hash changes or if files are added or removed, the application will exit with a status code of 1 to indicate there are changes. This action can be useful for scripting. For example, to run a nightly cron check of a manifest. A YAML based report will be generated as well.

**Tests** This application comes with tests. To run them, ensure you have `tox` installed (`pip install tox`). Then you can run `tox` to run the tests.

To build the docs, run `tox -e docs`. The HTML docs will be generated in the `.tox/docs/tmp/html/` directory.

**Faster YAML** By default, furtive will install and use the full Python implementation of the YAML parser which is very slow. In a testing environment, the Python implementation of the YAML loader took 1 minute to parse a 187,000 line furtive manifest file. By contrast, when the [LibYaml](#) parser was used, the loader took only 5 seconds to parse the same file.

To install the faster parser, perform the following steps:

1. Follow the [instructions from the LibYaml website](#) to download and install the latest release of libyaml.
2. Reinstall the PyYAML package by downloading the latest tar from the [PyYAML website](#) and running `python setup.py --with-libyaml install`

**CLI Reference** Command Line Interface (or Tool) reference.

Manage a Furtive manifest

```
usage: furtive [-h] [--basedir BASEDIR] [--manifest MANIFEST_PATH]
              [--log-level {debug,info,warn,error,critical}]
              [--exclude PATTERN] [--quiet] [--report-output FILE_NAME]
              [--version]
              {create,compare,check}
```

**Positional arguments:**

<b>action</b>	Which action to perform: <code>compare</code> - compare the current state of the files on the file system with the recorded state in the manifest file. Status code is 0 if the comparison was successful. <code>check</code> - check the integrity of files listed in the manifest. Same as <code>compare</code> but exits with status code 1 if there are changes to the files included in the manifest. That is, if any file hash changes or if files are added or removed, the application will exit with a status code of 1 to indicate there are changes. This action can be useful for
---------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

scripting. For example, to run a nightly cron check of a manifest. create  
 - create a new manifest from the files in the directory specified by the `--basedir` argument.

Possible choices: create, compare, check

### Options:

<b>--basedir=</b>	Directory containing files that will be checked. Default: .
<b>--manifest</b>	Location of the manifest file. Manifests may be located outside the directory indicated by <code>--basedir</code> . Must provide path and filename of the manifest file. Default: <code>&lt;basedir&gt;/manifest.yaml</code>
<b>--log-level=info</b>	verbosity of furtive  Possible choices: debug, info, warn, error, critical
<b>--exclude=[]</b>	Patterns to exclude files and directories from manifest. Can have multiple occurrences of this argument. Excludes are not stored in the manifest so it is up to the user to provide the same arguments every run. Patterns are evaluated as UNIX shell-style wildcard characters. See the [fnmatch documentation](https://docs.python.org/2/library/fnmatch.html) for more information.  It is important to note that exclusions are not stored. Therefore, they must be specified for every run of 'furtive'. Otherwise, the files which were previously excluded will be included and will show up as files added to the manifest.
<b>--quiet=False</b>	Only print out critical error messages. Do not print a report at the end of a compare run. Using this argument will override the log-level and set it to "critical". Only exceptions will be printed to terminal. The return code will be the only way to know if a Manifest has changed. This is useful for scripting such as a cron based manifest checks. Useful with the check command.
<b>--report-output=-</b>	File to print the diff report to. - for stdout. This can be consumed by other scripts to determine exactly what has changed within the manifest. Default: -
<b>--version</b>	show program's version number and exit

### API Reference

- *API Reference*
  - *Furtive Class*
  - *Sub-Modules*
    - \* *Hasher*
    - \* *Manifest*

This document is for developers of furtive, it contains the API functions

### Furtive Class Furtive - File Integrity Verification System

```
class furtive.Furtive(base_dir, manifest_path, exclude=None)
    Bases: object
```

Furtive is an application which stores file state and allows users to verify the state in the future. Example use cases include file archives and file transport.

If the manifest file exists, it will be automatically loaded. Calling `create()` will overwrite the existing manifest in memory as well as the file.

#### Parameters

- **base\_dir** (*str*) – Base directory to use for the manifest. Can be a full or relative path.
- **manifest\_path** (*str*) – Path to the manifest file. Can be a full or relative path.
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

#### `compare()`

Compare the hashes in the database with the current hashes of files on the file system.

**Returns** Dictionary of added, deleted, and changed files.

**Return type** dict

#### `create()`

Create and save a new manifest.

The contents of the new Manifest() will be saved to *manifest\_path*.

**Returns** None

## Sub-Modules

**Hasher** Manages the hashing of files

**class** `furtive.hasher.HashDirectory` (*directory*, *exclude=None*)

Bases: `object`

Object to manage hashing files in a directory.

This object is responsible for walking the directory tree and adding each file to a list. Once the directory walk has completed, each file path is passed to `hash_task()`. After each file has been hashed, this object will then create a Python dictionary of files with their associated hash.

#### Parameters

- **directory** (*str*) – Path to directory containing files
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

**Returns** Dictionary of file:hash

**Return type** dict

#### `excluded` (*file\_path*)

Should the file be excluded from the manifest?

Determines if a file should be excluded based on UNIX style pattern matching. Think `*`, `?`, and `[]` sequences.

For matchers, see <https://docs.python.org/2/library/fnmatch.html>

**Parameters** **file\_path** (*str*) – path of the file to match against.

**Returns** True or False indicating if the file should be excluded from the list of files contained within the manifest.

**Return type** bool

**hash\_files()**

Orchestrates the discovery and hashing of files.

Note: This method only supports the md5 hashing algorithm

`furtive.hasher.hash_task(file_path, hash_algorithm='md5')`

Responsible for hashing a file.

This function reads in the file `file_path` in small chunks the size of the hash algorithm's block size in order to avoid running out of memory. This means that this function should be able to read any file regardless of the size.

**Parameters**

- **file\_path** (*str*) – path of file to hash
- **hash\_algorithm** (*str*) – the hashing algorithm to use. All options available in *hashlib.algorithms* should work. See: <https://docs.python.org/2/library/hashlib.html>

**Returns** hash of file

**Return type** dict

`furtive.hasher.initializer(terminating_)`

Method to make terminating a global variable so that it is inherited by child processes.

**Manifest** Manifest of files and their hashes

**class** `furtive.manifest.Manifest(directory, manifest_file, exclude=None)`

Bases: object

Manifest of files and the associated hashes.

**Parameters**

- **directory** – directory which will serve as the root for the manifest. All files under the directory will be hashed and added to or compared with the manifest.
- **type** – str
- **manifest\_file** – file location of the manifest file. This is the path which will be used for the `create()` and `compare()` methods. If the file exists, the `create()` method will overwrite it.
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

**create()**

Creates a new manifest from the directory by calling `furtive.hasher.HashDirectory()` and placing the return dictionary in to *Manifest.manifest*.

**is\_empty()**

Determines if the manifest within memory is empty.

This simply checks to see if the manifest is None.

**Returns** True if manifest is empty, False otherwise.

**Return type** bool

**load()**

Load a manifest from the manifest file.

This method will open the manifest YAML file and load it in to the *manifest* object variable.



**save()**

Save the manifest to the manifest file.

Open a YAML file and dump the contents of the manifest to it.

## Requirements

- Python 2.7, 3.4, or 3.5

## Getting Started

To install furtive, run `pip install furtive`.

## CLI Usage

See the [CLI Reference](#) for more information about available command line arguments.

**Use Case Example** Suppose you have a million digital photos in a directory called `my-photos` that you have taken over the years. You would like to know if the files begin to decay due to hardware failure or something else. Alternatively, you may wish to have reassurance that your photos have not become corrupted while being stored in a cloud backup solution such as S3 or Glacier.

To record the current state of the files, run `furtive --basedir my-photos create`

This command creates the file `.manifest.yaml` in the `my-photos/` directory. The location and name of this file can be changed by using the `--manifest` argument.

At this point, you can be sure that you will know if a file has changed. To check the files on the file system to the recorded state in the manifest, run `furtive --basedir my-photos check`. The application will output a list of files which have been added, removed, or changed. This output is YAML format so it should be easy to parse. Additionally, furtive will exit with 1 indicating the check failed. This command can be placed in a cron job and setup to send a notification if a file has changed.

**Actions** There are a few actions which can be performed by furtive.

- **create** - create a new manifest from the files in the directory specified by the `--basedir` argument.
- **compare** - compare the current state of the files on the file system with the recorded state in the manifest file. A YAML based report will be created detailing which files have changed and which files have been added or removed. Status code is 0 if the comparison was successful.
- **check** - check the integrity of files listed in the manifest. Same as `compare` but exits with status code 1 if there are changes to the files included in the manifest. That is, if any file hash changes or if files are added or removed, the application will exit with a status code of 1 to indicate there are changes. This action can be useful for scripting. For example, to run a nightly cron check of a manifest. A YAML based report will be generated as well.

## Tests

This application comes with tests. To run them, ensure you have `tox` installed (`pip install tox`). Then you can run `tox` to run the tests.

To build the docs, run `tox -e docs`. The HTML docs will be generated in the `.tox/docs/tmp/html/` directory.

### Faster YAML

By default, furtive will install and use the full Python implementation of the YAML parser which is very slow. In a testing environment, the Python implementation of the YAML loader took 1 minute to parse a 187,000 line furtive manifest file. By contrast, when the [LibYaml](#) parser was used, the loader took only 5 seconds to parse the same file.

To install the faster parser, perform the following steps:

1. Follow the [instructions from the LibYaml website](#) to download and install the latest release of libyaml.
2. Reinstall the PyYAML package by downloading the latest tar from the [PyYAML website](#) and running `python setup.py --with-libyaml install`

### CLI Reference

Command Line Interface (or Tool) reference.

Manage a Furtive manifest

```
usage: furtive [-h] [--basedir BASEDIR] [--manifest MANIFEST_PATH]
              [--log-level {debug,info,warn,error,critical}]
              [--exclude PATTERN] [--quiet] [--report-output FILE_NAME]
              [--version]
              {create,compare,check}
```

#### Positional arguments:

<b>action</b>	Which action to perform: <code>compare</code> - compare the current state of the files on the file system with the recorded state in the manifest file. Status code is 0 if the comparison was successful. <code>check</code> - check the integrity of files listed in the manifest. Same as <code>compare</code> but exits with status code 1 if there are changes to the files included in the manifest. That is, if any file hash changes or if files are added or removed, the application will exit with a status code of 1 to indicate there are changes. This action can be useful for scripting. For example, to run a nightly cron check of a manifest. <code>create</code> - create a new manifest from the files in the directory specified by the <code>--basedir</code> argument.
---------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Possible choices: `create`, `compare`, `check`

#### Options:

<b>--basedir=.</b>	Directory containing files that will be checked. Default: <code>.</code>
<b>--manifest</b>	Location of the manifest file. Manifests may be located outside the directory indicated by <code>--basedir</code> . Must provide path and filename of the manifest file. Default: <code>&lt;basedir&gt;/manifest.yaml</code>
<b>--log-level=info</b>	verbosity of furtive  Possible choices: <code>debug</code> , <code>info</code> , <code>warn</code> , <code>error</code> , <code>critical</code>
<b>--exclude=[]</b>	Patterns to exclude files and directories from manifest. Can have multiple occurrences of this argument. Excludes are not stored in the manifest so it is up to the user to provide the same arguments every run. Patterns are

evaluated as UNIX shell-style wildcard characters. See the [fnmatch documentation](https://docs.python.org/2/library/fnmatch.html) for more information.

It is important to note that exclusions are not stored. Therefore, they must be specified for every run of 'furtive'. Otherwise, the files which were previously excluded will be included and will show up as files added to the manifest.

<b>--quiet=False</b>	Only print out critical error messages. Do not print a report at the end of a compare run. Using this argument will override the log-level and set it to "critical". Only exceptions will be printed to terminal. The return code will be the only way to know if a Manifest has changed. This is useful for scripting such as a cron based manifest checks. Useful with the check command.
<b>--report-output=-</b>	File to print the diff report to. - for stdout. This can be consumed by other scripts to determine exactly what has changed within the manifest. Default: -
<b>--version</b>	show program's version number and exit

## API Reference

- *API Reference*
  - *Furtive Class*
  - *Sub-Modules*
    - \* *Hasher*
    - \* *Manifest*

This document is for developers of furtive, it contains the API functions

## Furtive Class

Furtive - File Integrity Verification System

**class** `furtive.Furtive` (*base\_dir*, *manifest\_path*, *exclude=None*)

Bases: `object`

Furtive is an application which stores file state and allows users to verify the state in the future. Example use cases include file archives and file transport.

If the manifest file exists, it will be automatically loaded. Calling `create()` will overwrite the existing manifest in memory as well as the file.

### Parameters

- **base\_dir** (*str*) – Base directory to use for the manifest. Can be a full or relative path.
- **manifest\_path** (*str*) – Path to the manifest file. Can be a full or relative path.
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

**compare** ()

Compare the hashes in the database with the current hashes of files on the file system.

**Returns** Dictionary of added, deleted, and changed files.

**Return type** dict

**create()**

Create and save a new manifest.

The contents of the new Manifest() will be saved to *manifest\_path*.

**Returns** None

**Sub-Modules**

**Hasher** Manages the hashing of files

**class** furtive.hasher.**HashDirectory**(*directory*, *exclude=None*)

Bases: object

Object to manage hashing files in a directory.

This object is responsible for walking the directory tree and adding each file to a list. Once the directory walk has completed, each file path is passed to `hash_task()`. After each file has been hashed, this object will then create a Python dictionary of files with their associated hash.

**Parameters**

- **directory** (*str*) – Path to directory containing files
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

**Returns** Dictionary of file:hash

**Return type** dict

**excluded**(*file\_path*)

Should the file be excluded from the manifest?

Determines if a file should be excluded based on UNIX style pattern matching. Think `*`, `?`, and `[]` sequences.

For matchers, see <https://docs.python.org/2/library/fnmatch.html>

**Parameters** **file\_path** (*str*) – path of the file to match against.

**Returns** True or False indicating if the file should be excluded from the list of files contained within the manifest.

**Return type** bool

**hash\_files()**

Orchestrates the discovery and hashing of files.

Note: This method only supports the md5 hashing algorithm

**furtive.hasher.hash\_task**(*file\_path*, *hash\_algorithm='md5'*)

Responsible for hashing a file.

This function reads in the file *file\_path* in small chunks the size of the hash algorithm's block size in order to avoid running out of memory. This means that this function should be able to read any file regardless of the size.

**Parameters**

- **file\_path** (*str*) – path of file to hash
- **hash\_algorithm** (*str*) – the hashing algorithm to use. All options available in *hashlib* should work. See: <https://docs.python.org/2/library/hashlib.html>

**Returns** hash of file

**Return type** dict

`furtive.hasher.initializer(terminating_)`

Method to make terminating a global variable so that it is inherited by child processes.

**Manifest** Manifest of files and their hashes

**class** `furtive.manifest.Manifest(directory, manifest_file, exclude=None)`

Bases: object

Manifest of files and the associated hashes.

#### Parameters

- **directory** – directory which will serve as the root for the manifest. All files under the directory will be hashed and added to or compared with the manifest.
- **type** – str
- **manifest\_file** – file location of the manifest file. This is the path which will be used for the `create()` and `compare()` methods. If the file exists, the `create()` method will overwrite it.
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

**create()**

Creates a new manifest from the directory by calling `furtive.hasher.HashDirectory()` and placing the return dictionary in to *Manifest.manifest*.

**is\_empty()**

Determines if the manifest within memory is empty.

This simply checks to see if the manifest is None.

**Returns** True if manifest is empty, False otherwise.

**Return type** bool

**load()**

Load a manifest from the manifest file.

This method will open the manifest YAML file and load it in to the *manifest* object variable.

**save()**

Save the manifest to the manifest file.

Open a YAML file and dump the contents of the manifest to it.

## 1.1.2 Requirements

- Python 2.7, 3.4, or 3.5

## 1.1.3 Getting Started

To install furtive, run `pip install furtive`.

## 1.1.4 CLI Usage

See the [CLI Reference](#) for more information about available command line arguments.

## Use Case Example

Suppose you have a million digital photos in a directory called `my-photos` that you have taken over the years. You would like to know if the files begin to decay due to hardware failure or something else. Alternatively, you may wish to have reassurance that your photos have not become corrupted while being stored in a cloud backup solution such as S3 or Glacier.

To record the current state of the files, run `furtive --basedir my-photos create`

This command creates the file `.manifest.yaml` in the `my-photos/` directory. The location and name of this file can be changed by using the `--manifest` argument.

At this point, you can be sure that you will know if a file has changed. To check the files on the file system to the recorded state in the manifest, run `furtive --basedir my-photos check`. The application will output a list of files which have been added, removed, or changed. This output is YAML format so it should be easy to parse. Additionally, `furtive` will exit with 1 indicating the check failed. This command can be placed in a cron job and setup to send a notification if a file has changed.

## Actions

There are a few actions which can be performed by `furtive`.

- **create** - create a new manifest from the files in the directory specified by the `--basedir` argument.
- **compare** - compare the current state of the files on the file system with the recorded state in the manifest file. A YAML based report will be created detailing which files have changed and which files have been added or removed. Status code is 0 if the comparison was successful.
- **check** - check the integrity of files listed in the manifest. Same as `compare` but exits with status code 1 if there are changes to the files included in the manifest. That is, if any file hash changes or if files are added or removed, the application will exit with a status code of 1 to indicate there are changes. This action can be useful for scripting. For example, to run a nightly cron check of a manifest. A YAML based report will be generated as well.

### 1.1.5 Tests

This application comes with tests. To run them, ensure you have `tox` installed (`pip install tox`). Then you can run `tox` to run the tests.

To build the docs, run `tox -e docs`. The HTML docs will be generated in the `.tox/docs/tmp/html/` directory.

### 1.1.6 Faster YAML

By default, `furtive` will install and use the full Python implementation of the YAML parser which is very slow. In a testing environment, the Python implementation of the YAML loader took 1 minute to parse a 187,000 line `furtive` manifest file. By contrast, when the [LibYaml](#) parser was used, the loader took only 5 seconds to parse the same file.

To install the faster parser, perform the following steps:

1. Follow the [instructions from the LibYaml website](#) to download and install the latest release of `libyaml`.
2. Reinstall the `PyYAML` package by downloading the latest tar from the [PyYAML website](#) and running `python setup.py --with-libyaml install`

## 1.2 CLI Reference

Command Line Interface (or Tool) reference.

Manage a Furtive manifest

```
usage: furtive [-h] [--basedir BASEDIR] [--manifest MANIFEST_PATH]
              [--log-level {debug,info,warn,error,critical}]
              [--exclude PATTERN] [--quiet] [--report-output FILE_NAME]
              [--version]
              {create,compare,check}
```

**Positional arguments:**

<b>action</b>	Which action to perform: compare - compare the current state of the files on the file system with the recorded state in the manifest file. Status code is 0 if the comparison was successful. check - check the integrity of files listed in the manifest. Same as compare but exits with status code 1 if there are changes to the files included in the manifest. That is, if any file hash changes or if files are added or removed, the application will exit with a status code of 1 to indicate there are changes. This action can be useful for scripting. For example, to run a nightly cron check of a manifest. create - create a new manifest from the files in the directory specified by the --basedir argument.
---------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Possible choices: create, compare, check

**Options:**

<b>--basedir=.</b>	Directory containing files that will be checked. Default: .
<b>--manifest</b>	Location of the manifest file. Manifests may be located outside the directory indicated by --basedir. Must provide path and filename of the manifest file. Default: <basedir>/manifest.yaml
<b>--log-level=info</b>	verbosity of furtive  Possible choices: debug, info, warn, error, critical
<b>--exclude=[]</b>	Patterns to exclude files and directories from manifest. Can have multiple occurrences of this argument. Excludes are not stored in the manifest so it is up to the user to provide the same arguments every run. Patterns are evaluated as UNIX shell-style wildcard characters. See the [fnmatch documentation](https://docs.python.org/2/library/fnmatch.html) for more information.  It is important to note that exclusions are not stored. Therefore, they must be specified for every run of 'furtive'. Otherwise, the files which were previously excluded will be included and will show up as files added to the manifest.
<b>--quiet=False</b>	Only print out critical error messages. Do not print a report at the end of a compare run. Using this argument will override the log-level and set it to "critical". Only exceptions will be printed to terminal. The return code will be the only way to know if a Manifest has changed. This is useful for scripting such as a cron based manifest checks. Useful with the check command.
<b>--report-output=-</b>	File to print the diff report to. - for stdout. This can be consumed by other scripts to determine exactly what has changed within the manifest. Default: -

**--version** show program's version number and exit

## 1.3 API Reference

- *API Reference*
  - *Furtive Class*
  - *Sub-Modules*
    - \* *Hasher*
    - \* *Manifest*

This document is for developers of furtive, it contains the API functions

### 1.3.1 Furtive Class

Furtive - File Integrity Verification System

**class** `furtive.Furtive` (*base\_dir*, *manifest\_path*, *exclude=None*)

Bases: `object`

Furtive is an application which stores file state and allows users to verify the state in the future. Example use cases include file archives and file transport.

If the manifest file exists, it will be automatically loaded. Calling `create()` will overwrite the existing manifest in memory as well as the file.

#### Parameters

- **base\_dir** (*str*) – Base directory to use for the manifest. Can be a full or relative path.
- **manifest\_path** (*str*) – Path to the manifest file. Can be a full or relative path.
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

**compare** ()

Compare the hashes in the database with the current hashes of files on the file system.

**Returns** Dictionary of added, deleted, and changed files.

**Return type** dict

**create** ()

Create and save a new manifest.

The contents of the new Manifest() will be saved to *manifest\_path*.

**Returns** None

### 1.3.2 Sub-Modules

#### Hasher

Manages the hashing of files



**class** `furtive.hasher.HashDirectory` (*directory*, *exclude=None*)

Bases: `object`

Object to manage hashing files in a directory.

This object is responsible for walking the directory tree and adding each file to a list. Once the directory walk has completed, each file path is passed to `hash_task()`. After each file has been hashed, this object will then create a Python dictionary of files with their associated hash.

#### Parameters

- **directory** (*str*) – Path to directory containing files
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

**Returns** Dictionary of file:hash

**Return type** `dict`

**excluded** (*file\_path*)

Should the file be excluded from the manifest?

Determines if a file should be excluded based on UNIX style pattern matching. Think `*`, `?`, and `[]` sequences.

For matchers, see <https://docs.python.org/2/library/fnmatch.html>

**Parameters** **file\_path** (*str*) – path of the file to match against.

**Returns** True or False indicating if the file should be excluded from the list of files contained within the manifest.

**Return type** `bool`

**hash\_files** ()

Orchestrates the discovery and hashing of files.

Note: This method only supports the md5 hashing algorithm

`furtive.hasher.hash_task` (*file\_path*, *hash\_algorithm='md5'*)

Responsible for hashing a file.

This function reads in the file *file\_path* in small chunks the size of the hash algorithm's block size in order to avoid running out of memory. This means that this function should be able to read any file regardless of the size.

#### Parameters

- **file\_path** (*str*) – path of file to hash
- **hash\_algorithm** (*str*) – the hashing algorithm to use. All options available in *hashlib.algorithms* should work. See: <https://docs.python.org/2/library/hashlib.html>

**Returns** hash of file

**Return type** `dict`

`furtive.hasher.initializer` (*terminating\_*)

Method to make terminating a global variable so that it is inherited by child processes.

## Manifest

Manifest of files and their hashes

**class** `furtive.manifest.Manifest` (*directory*, *manifest\_file*, *exclude=None*)

Bases: `object`

Manifest of files and the associated hashes.

**Parameters**

- **directory** – directory which will serve as the root for the manifest. All files under the directory will be hashed and added to or compared with the manifest.
- **type** – str
- **manifest\_file** – file location of the manifest file. This is the path which will be used for the `create()` and `compare()` methods. If the file exists, the `create()` method will overwrite it.
- **exclude** (*list*) – list containing patterns to use to exclude files from the manifest.

**create()**

Creates a new manifest from the directory by calling `furtive.hasher.HashDirectory()` and placing the return dictionary in to *Manifest.manifest*.

**is\_empty()**

Determines if the manifest within memory is empty.

This simply checks to see if the manifest is `None`.

**Returns** True if manifest is empty, False otherwise.

**Return type** bool

**load()**

Load a manifest from the manifest file.

This method will open the manifest YAML file and load it in to the *manifest* object variable.

**save()**

Save the manifest to the manifest file.

Open a YAML file and dump the contents of the manifest to it.

---

## Requirements

---

- Python 2.7, 3.4, or 3.5



---

## Getting Started

---

To install `furtive`, run `pip install furtive`.



---

## CLI Usage

---

See the [CLI Reference](#) for more information about available command line arguments.

### 4.1 Use Case Example

Suppose you have a million digital photos in a directory called `my-photos` that you have taken over the years. You would like to know if the files begin to decay due to hardware failure or something else. Alternatively, you may wish to have reassurance that your photos have not become corrupted while being stored in a cloud backup solution such as S3 or Glacier.

To record the current state of the files, run `furtive --basedir my-photos create`

This command creates the file `.manifest.yaml` in the `my-photos/` directory. The location and name of this file can be changed by using the `--manifest` argument.

At this point, you can be sure that you will know if a file has changed. To check the files on the file system to the recorded state in the manifest, run `furtive --basedir my-photos check`. The application will output a list of files which have been added, removed, or changed. This output is YAML format so it should be easy to parse. Additionally, `furtive` will exit with 1 indicating the check failed. This command can be placed in a cron job and setup to send a notification if a file has changed.

### 4.2 Actions

There are a few actions which can be performed by `furtive`.

- **create** - create a new manifest from the files in the directory specified by the `--basedir` argument.
- **compare** - compare the current state of the files on the file system with the recorded state in the manifest file. A YAML based report will be created detailing which files have changed and which files have been added or removed. Status code is 0 if the comparison was successful.
- **check** - check the integrity of files listed in the manifest. Same as `compare` but exits with status code 1 if there are changes to the files included in the manifest. That is, if any file hash changes or if files are added or removed, the application will exit with a status code of 1 to indicate there are changes. This action can be useful for scripting. For example, to run a nightly cron check of a manifest. A YAML based report will be generated as well.





---

### Tests

---

This application comes with tests. To run them, ensure you have `tox` installed (`pip install tox`). Then you can run `tox` to run the tests.

To build the docs, run `tox -e docs`. The HTML docs will be generated in the `.tox/docs/tmp/html/` directory.



---

## Faster YAML

---

By default, `furtive` will install and use the full Python implementation of the YAML parser which is very slow. In a testing environment, the Python implementation of the YAML loader took 1 minute to parse a 187,000 line `furtive` manifest file. By contrast, when the [LibYaml](#) parser was used, the loader took only 5 seconds to parse the same file.

To install the faster parser, perform the following steps:

1. Follow the [instructions from the LibYaml website](#) to download and install the latest release of `libyaml`.
2. Reinstall the `PyYAML` package by downloading the latest tar from the [PyYAML website](#) and running `python setup.py --with-libyaml install`



**f**

`furtive`, [60](#)

`furtive.hasher`, [60](#)

`furtive.manifest`, [61](#)



## C

compare() (furtive.Furtive method), 6, 11, 15, 20, 24, 29, 33, 37, 42, 46, 51, 55, 60  
create() (furtive.Furtive method), 6, 11, 15, 20, 24, 29, 33, 38, 42, 46, 51, 55, 60  
create() (furtive.manifest.Manifest method), 8, 12, 17, 21, 26, 30, 34, 39, 43, 48, 52, 57, 62

## E

excluded() (furtive.hasher.HashDirectory method), 7, 11, 16, 20, 25, 29, 34, 38, 42, 47, 51, 56, 61

## F

Furtive (class in furtive), 6, 11, 15, 19, 24, 28, 33, 37, 42, 46, 50, 55, 60  
furtive (module), 6, 11, 15, 19, 24, 28, 33, 37, 42, 46, 50, 55, 60  
furtive.hasher (module), 7, 11, 16, 20, 24, 29, 33, 38, 42, 47, 51, 56, 60  
furtive.manifest (module), 8, 12, 17, 21, 25, 30, 34, 39, 43, 48, 52, 57, 61

## H

hash\_files() (furtive.hasher.HashDirectory method), 7, 12, 16, 20, 25, 29, 34, 38, 43, 47, 51, 56, 61  
hash\_task() (in module furtive.hasher), 7, 12, 16, 21, 25, 29, 34, 38, 43, 47, 52, 56, 61  
HashDirectory (class in furtive.hasher), 7, 11, 16, 20, 24, 29, 33, 38, 42, 47, 51, 56, 60

## I

initializer() (in module furtive.hasher), 8, 12, 16, 21, 25, 30, 34, 39, 43, 47, 52, 57, 61  
is\_empty() (furtive.manifest.Manifest method), 8, 12, 17, 21, 26, 30, 35, 39, 43, 48, 52, 57, 62

## L

load() (furtive.manifest.Manifest method), 8, 13, 17, 21, 26, 30, 35, 39, 44, 48, 52, 57, 62

## M

Manifest (class in furtive.manifest), 8, 12, 17, 21, 25, 30, 34, 39, 43, 48, 52, 57, 61

## S

save() (furtive.manifest.Manifest method), 8, 13, 17, 21, 26, 30, 35, 39, 44, 48, 52, 57, 62