organize Documentation

Release 1.5.2

Thomas Feldmann

Jul 29, 2019

Contents

organize is a command line utility to automate file organization tasks.

http://github.com/tfeldmann/organize

CHAPTER 1

Contents:

1.1 Quickstart

1.1.1 Installation

Requirements: Python 3.3+

organize is installed via pip:

On macOS / Windows: `$ pip3 install organize-tool`

On Linux: `$ sudo pip3 install organize-tool`

1.1.2 Creating your first config file

To edit the configuration in your $EDITOR, run:

```
$ organize config
```

For example your configuration file could look like this:

Listing 1: config.yaml

```
rules:
  # move screenshots into "Screenshots" folder
  - folders:
      - ~/Desktop
    filters:
      - filename:
          startswith: Screen Shot
    actions:
      - move: ~/Desktop/Screenshots/

  # move incomplete downloads older > 30 days into the trash
```

(continues on next page)

(continued from previous page)

```
    - folders:
        - ~/Downloads
      filters:
        - extension:
            - crdownload
            - part
            - download
        - lastmodified:
            days: 30
      actions:
        - trash
```

Note: You can run `$ organize config --path` to show the full path to the configuration file.

1.1.3 Simulate and run

After you saved the configuration file, run `$ organize sim` to show a simulation of how your files would be organized.

If you like what you see, run `$ organize run` to organize your files.

Note: Congrats! You just automated some tedious cleaning tasks! Continue to *Configuration* to see the full potential of organize or skip directly to the *Filters* and *Actions*.

1.2 Configuration

1.2.1 Editing the configuration

All configuration takes place in your *config.yaml* file.

- To edit your configuration in `$EDITOR` run:

  ```
  $ organize config  # example: "EDITOR=vim organize config"
  ```

- To show the full path to your configuration file:

  ```
  $ organize config --path
  ```

- To open the folder containing the configuration file:

  ```
  $ organize config --open-folder
  ```

- To debug your configuration run:

  ```
  $ organize config --debug
  ```

1.2.2 Environment variables

- `$EDITOR` - The editor used to edit the config file.
- `$ORGANIZE_CONFIG` - The config file path. Is overridden by `--config-file` cmd line argument.

1.2.3 Rule syntax

The rule configuration is done in YAML. You need a top-level element `rules` which contains a list of rules. Each rule defines `folders`, `filters` (optional) and `actions`.

Listing 2: config.yaml

```yaml
rules:
  - folders:
      - ~/Desktop
      - /some/folder/
    filters:
      - lastmodified:
          days: 40
          mode: newer
      - extension: pdf
    actions:
      - move: ~/Desktop/Target/
      - trash

  - folders:
      - ~/Inbox
    filters:
      - extension: pdf
    actions:
      - move: ~/otherinbox
    # optional settings:
    enabled: true
    subfolders: true
    system_files: false
```

- `folders` is a list of folders you want to organize.
- `filters` is a list of filters to apply to the files - you can filter by file extension, last modified date, regular expressions and many more. See *Filters*.
- `actions` is a list of actions to apply to the filtered files. You can put them into the trash, move them into another folder and many more. See *Actions*.

Other optional per rule settings:

- `enabled` can be used to temporarily disable single rules. Default = true
- `subfolders` specifies whether subfolders should be included in the search. Default = false. This setting only applies to folders without glob wildcards.
- `system_files` specifies whether to include system files (desktop.ini, thumbs.db, .DS_Store) in the search. Default = false

1.2.4 Folder syntax

Every rule in your configuration file needs to know the folders it applies to. The easiest way is to define the rules like this:

Listing 3: config.yaml

```yaml
rules:
  - folders:
      - /path/one
      - /path/two
    filters: ...
    actions: ...

  - folders:
      - /path/one
      - /another/path
    filters: ...
    actions: ...
```

Globstrings

You can use globstrings in the folder lists. For example to get all files with filenames ending with `_ui` and any file extension you can use:

Listing 4: config.yaml

```yaml
rules:
  - folders:
      - '~/Downloads/*_ui.*'
    actions:
      - echo: '{path}'
```

You can use globstrings to recurse through subdirectories (alternatively you can use the `subfolders: true` setting as shown below)

Listing 5: config.yaml

```yaml
rules:
  - folders:
      - '~/Downloads/**/*.*'
    actions:
      - echo: 'base {basedir}, path {path}, relative: {relative_path}'

  # alternative syntax
  - folders:
      - ~/Downloads
    subfolders: true
    actions:
      - echo: 'base {basedir}, path {path}, relative: {relative_path}'
```

The following example recurses through all subdirectories in your downloads folder and finds files with ending in `.c` and `.h`.

Listing 6: config.yaml

```yaml
rules:
  - folders:
      - '~/Downloads/**/*.[c|h]'
    actions:
      - echo: '{path}'
```

Note:

- You have to target files with the globstring, not folders. So to scan through all folders starting with *log_* you would write `yourpath/log_*/*`

Excluding files and folders

Files and folders can be excluded by prepending an exclamation mark. The following example selects all files in `~/Downloads` and its subfolders - excluding the folder `Software`:

Listing 7: config.yaml

```yaml
rules:
  - folders:
      - '~/Downloads/**/*'
      - '! ~/Downloads/Software'
    actions:
      - echo: '{path}'
```

Globstrings can be used to exclude only specific files / folders. This example:

- adds all files in `~/Downloads`
- exludes files from that list whose name contains the word `system` ending in `.bak`
- adds all files from `~/Documents`
- excludes the file `~/Documents/important.txt`.

Listing 8: config.yaml

```yaml
rules:
  - folders:
      - '~/Downloads/**/*'
      - '! ~/Downloads/**/*system*.bak'
      - '~/Documents'
      - '! ~/Documents/important.txt'
    actions:
      - echo: '{path}'
```

Note:

- Files and folders are included and excluded in the order you specify them!
- Please make sure your are putting the exclamation mark within quotation marks.

Aliases

Instead of repeating the same folders in each and every rule you can use an alias for multiple folders which you can then reference in each rule. Aliases are a standard feature of the YAML syntax.

Listing 9: config.yaml

```yaml
all_my_messy_folders: &all
  - ~/Desktop
  - ~/Downloads
  - ~/Documents
  - ~/Dropbox

rules:
  - folders: *all
    filters: ...
    actions: ...

  - folders: *all
    filters: ...
    actions: ...
```

You can even use multiple folder lists:

Listing 10: config.yaml

```yaml
private_folders: &private
  - '/path/private'
  - '~/path/private'

work_folders: &work
  - '/path/work'
  - '~/My work folder'

all_folders: &all
  - *private
  - *work

rules:
  - folders: *private
    filters: ...
    actions: ...

  - folders: *work
    filters: ...
    actions: ...

  - folders: *all
    filters: ...
    actions: ...

  # same as *all
  - folders:
      - *work
      - *private
    filters: ...
    actions: ...
```

1.2.5 Filter syntax

`filters` is a list of *Filters*. Filters are defined like this:

Listing 11: config.yaml

```yaml
rules:
  - folders: ...
    actions: ...
    filters:
      # filter without parameters
      - FilterName

      # filter with a single parameter
      - FilterName: parameter

      # filter expecting a list as parameter
      - FilterName:
        - first
        - second
        - third

      # filter with multiple parameters
      - FilterName:
          parameter1: true
          option2: 10.51
          third_argument: test string
```

Note: Every filter comes with multiple usage examples which should be easy to adapt for your use case!

1.2.6 Action syntax

`actions` is a list of *Actions*. Actions can be defined like this:

Listing 12: config.yaml

```yaml
rules:
  - folders: ...
    actions:
      # action without parameters
      - ActionName

      # action with a single parameter
      - ActionName: parameter

      # filter with multiple parameters
      - ActionName:
          parameter1: true
          option2: 10.51
          third_argument: test string
```

Note: Every action comes with multiple usage examples which should be easy to adapt for your use case!

Variable substitution (placeholders)

You can use placeholder variables in your actions.

Placeholder variables are used with curly braces `{var}`. You always have access to the variables `{path}`, `{basedir}` and `{relative_path}`:

- `{path}` – is the full path to the current file
- `{basedir}` – the current base folder (the base folder is the folder you specify in your configuration).
- `{relative_path}` – the relative path from `{basedir}` to `{path}`

Use the dot notation to access properties of `{path}`, `{basedir}` and `{relative_path}`:

- `{path}` – the full path to the current file
- `{path.name}` – the full filename including extension
- `{path.stem}` – just the file name without extension
- `{path.suffix}` – the file extension
- `{path.parent}` – the parent folder of the current file
- `{path.parent.parent}` – parent calls are chainable…
- `{basedir}` – the full path to the current base folder
- `{basedir.parent}` – the full path to the base folder's parent

and any other property of the python `pathlib.Path` (official documentation) object.

Additionally *Filters* may emit placeholder variables when applied to a path. Check the documentation and examples of the filter to see available placeholder variables and usage examples.

Some examples include:

- `{lastmodified.year}` – the year the file was last modified
- `{regex.yournamedgroup}` – anything you can extract via regular expressions
- `{extension.upper}` – the file extension in uppercase
- … and many more.

1.3 Filters

1.3.1 Extension

class Extension(**extensions*)

Filter by file extension

Parameters **extensions** – The file extensions to match (does not need to start with a colon).

Returns

- `{extension}` – the original file extension (without colon)
- `{extension.lower}` – the file extension in lowercase
- `{extension.upper}` – the file extension in UPPERCASE

Examples:

- Match a single file extension:

Listing 13: config.yaml

```yaml
rules:
  - folders: '~/Desktop'
    filters:
      - extension: png
    actions:
      - echo: 'Found PNG file: {path}'
```

- Match multiple file extensions:

Listing 14: config.yaml

```yaml
rules:
  - folders: '~/Desktop'
    filters:
      - extension:
        - .jpg
        - jpeg
    actions:
      - echo: 'Found JPG file: {path}'
```

- Make all file extensions lowercase:

Listing 15: config.yaml

```yaml
rules:
  - folder: '~/Desktop'
    filters:
      - Extension
    actions:
      - rename: '{path.stem}.{extension.lower}'
```

- Using extension lists:

Listing 16: config.yaml

```yaml
img_ext: &img
  - png
  - jpg
  - tiff

audio_ext: &audio
  - mp3
  - wav
  - ogg

rules:
  - folders: '~/Desktop'
    filters:
      - extension:
        - *img
        - *audio
    actions:
      - echo: 'Found media file: {path}'
```

1.3.2 Filename

class Filename(*startswith=”, contains=”, endswith=”, case_sensitive=True*)

Match files by filename

Parameters

- **startswith** (*str*) – The filename must begin with the given string
- **contains** (*str*) – The filename must contain the given string
- **endswith** (*str*) – The filename (without extension) must end with the given string
- **case_sensitive = True** (*bool*) – By default, the matching is case sensitive. Change this to False to use case insensitive matching.

Examples:

- Match all files starting with ‘Invoice’:

Listing 17: config.yaml

```yaml
rules:
  - folders: '~/Desktop'
    filters:
      - filename:
          startswith: Invoice
    actions:
      - echo: 'This is an invoice'
```

- Match all files starting with ‘A’ end containing the string ‘hole’ (case insensitive)

Listing 18: config.yaml

```yaml
rules:
  - folders: '~/Desktop'
    filters:
      - filename:
          startswith: A
          contains: hole
          case_sensitive: false
    actions:
      - echo: 'Found a match.'
```

1.3.3 LastModified

class LastModified(*days=0, hours=0, minutes=0, seconds=0, mode='older'*)

Matches files by last modified date

Parameters

- **days** (*int*) – specify number of days
- **hours** (*int*) – specify number of hours
- **minutes** (*int*) – specify number of minutes
- **mode** (*str*) – either ‘older’ or ‘newer’. ‘older’ matches all files last modified before the given time, ‘newer’ matches all files last modified within the given time. (default = ‘older’)

Returns

- `{lastmodified.year}` – the year the file was last modified
- `{lastmodified.month}` – the month the file was last modified
- `{lastmodified.day}` – the day the file was last modified
- `{lastmodified.hour}` – the hour the file was last modified
- `{lastmodified.minute}` – the minute the file was last modified
- `{lastmodified.second}` – the second the file was last modified

Examples:

- Show all files on your desktop last modified at least 10 days ago:

Listing 19: config.yaml

```yaml
rules:
  - folders: '~/Desktop'
    filters:
      - lastmodified:
          days: 10
    actions:
      - echo: 'Was modified at least 10 days ago'
```

- Show all files on your desktop which were modified within the last 5 hours:

Listing 20: config.yaml

```yaml
rules:
  - folders: '~/Desktop'
    filters:
      - lastmodified:
          hours: 5
          mode: newer
    actions:
      - echo: 'Was modified within the last 5 hours'
```

- Sort pdfs by year of last modification

Listing 21: config.yaml

```yaml
rules:
  - folders: '~/Documents'
    filters:
      - extension: pdf
      - LastModified
    actions:
      - move: '~/Documents/PDF/{lastmodified.year}/'
```

1.3.4 Created

class Created(*days=0*, *hours=0*, *minutes=0*, *seconds=0*, *mode='older'*)

Matches files by created date

Parameters

- **days** (*int*) – specify number of days

- **hours** (*int*) – specify number of hours
- **minutes** (*int*) – specify number of minutes
- **mode** (*str*) – either 'older' or 'newer'. 'older' matches all files created before the given time, 'newer' matches all files created within the given time. (default = 'older')

Returns

- `{created.year}` – the year the file was created
- `{created.month}` – the month the file was created
- `{created.day}` – the day the file was created
- `{created.hour}` – the hour the file was created
- `{created.minute}` – the minute the file was created
- `{created.second}` – the second the file was created

Examples:

- Show all files on your desktop created at least 10 days ago:

Listing 22: config.yaml

```yaml
rules:
  - folders: '~/Desktop'
    filters:
      - created:
          days: 10
    actions:
      - echo: 'Was created at least 10 days ago'
```

- Show all files on your desktop which were created within the last 5 hours:

Listing 23: config.yaml

```yaml
rules:
  - folders: '~/Desktop'
    filters:
      - created:
          hours: 5
          mode: newer
    actions:
      - echo: 'Was created within the last 5 hours'
```

- Sort pdfs by year of creation:

Listing 24: config.yaml

```yaml
rules:
  - folders: '~/Documents'
    filters:
      - extension: pdf
      - created
    actions:
      - move: '~/Documents/PDF/{created.year}/'
```

1.3.5 Regex

class Regex(*expr*)

Matches filenames with the given regular expression

Parameters **expr** (*str*) – The regular expression to be matched.

Any named groups in your regular expression will be returned like this:

Returns

- `{regex.yourgroupname}` – The text matched with the named group `(?P<yourgroupname>)`

Examples:

- Match an invoice with a regular expression:

Listing 25: config.yaml

```yaml
rules:
  - folders: '~/Desktop'
    filters:
      - regex: '^RG(\d{12})-sig\.pdf$'
    actions:
      - move: '~/Documents/Invoices/1und1/'
```

- Match and extract data from filenames with regex named groups: This is just like the previous example but we rename the invoice using the invoice number extracted via the regular expression and the named group `the_number`.

Listing 26: config.yaml

```yaml
rules:
  - folders: ~/Desktop
    filters:
      - regex: '^RG(?P<the_number>\d{12})-sig\.pdf$'
    actions:
      - move: ~/Documents/Invoices/1und1/{regex.the_number}.pdf
```

1.4 Actions

1.4.1 Move

class Move(*dest*[, *overwrite=False*][, *counter_separator=' '*])

Move a file to a new location. The file can also be renamed. If the specified path does not exist it will be created.

If you only want to rename the file and keep the folder, it is easier to use the Rename-Action.

Parameters

- **dest** (*str*) – The destination folder or path. If *dest* ends with a slash / backslash, the file will be moved into this folder and not renamed.
- **overwrite** (*bool*) – specifies whether existing files should be overwritten. Otherwise it will start enumerating files (append a counter to the filename) to resolve naming conflicts. [Default: False]

- **counter_separator** (*str*) – specifies the separator between filename and the appended counter. Only relevant if **overwrite** is disabled. [Default: `' '`]

Examples:

- Move all pdfs and jpgs from the desktop into the folder "~/Desktop/media/". Filenames are not changed.

Listing 27: config.yaml

```yaml
rules:
  - folders: ~/Desktop
    filters:
      - extension:
          - pdf
          - jpg
    actions:
      - move: '~/Desktop/media/'
```

- Use a placeholder to move all .pdf files into a "PDF" folder and all .jpg files into a "JPG" folder. Existing files will be overwritten.

Listing 28: config.yaml

```yaml
rules:
  - folders: ~/Desktop
    filters:
      - extension:
          - pdf
          - jpg
    actions:
      - move:
          dest: '~/Desktop/{extension.upper}/'
          overwrite: true
```

- Move pdfs into the folder *Invoices*. Keep the filename but do not overwrite existing files. To prevent overwriting files, an index is added to the filename, so `somefile.jpg` becomes `somefile 2.jpg`.

Listing 29: config.yaml

```yaml
rules:
  - folders: ~/Desktop/Invoices
    filters:
      - extension:
          - pdf
    actions:
      - move:
          dest: '~/Documents/Invoices/'
          overwrite: false
          counter_separator: '_'
```

1.4.2 Copy

class Copy(*dest*[, *overwrite=False*][, *counter_separator=' '*])
Copy a file to a new location. If the specified path does not exist it will be created.

Parameters

- **dest** (*str*) – The destination where the file should be copied to. If *dest* ends with a slash / backslash, the file will be copied into this folder and keep its original name.
- **overwrite** (*bool*) – specifies whether existing files should be overwritten. Otherwise it will start enumerating files (append a counter to the filename) to resolve naming conflicts. [Default: False]
- **counter_separator** (*str*) – specifies the separator between filename and the appended counter. Only relevant if **overwrite** is disabled. [Default: `' '`]

Examples:

- Copy all pdfs into *~/Desktop/somefolder/* and keep filenames

Listing 30: config.yaml

```yaml
rules:
  - folders: ~/Desktop
    filters:
      - extension: pdf
    actions:
      - copy: '~/Desktop/somefolder/'
```

- Use a placeholder to copy all .pdf files into a "PDF" folder and all .jpg files into a "JPG" folder. Existing files will be overwritten.

Listing 31: config.yaml

```yaml
rules:
  - folders: ~/Desktop
    filters:
      - extension:
          - pdf
          - jpg
    actions:
      - copy:
          dest: '~/Desktop/{extension.upper}/'
          overwrite: true
```

- Copy into the folder *Invoices*. Keep the filename but do not overwrite existing files. To prevent overwriting files, an index is added to the filename, so *somefile.jpg* becomes *somefile 2.jpg*. The counter separator is ' ' by default, but can be changed using the *counter_separator* property.

Listing 32: config.yaml

```yaml
rules:
  - folders: ~/Desktop/Invoices
    filters:
      - extension:
          - pdf
    actions:
      - copy:
          dest: '~/Documents/Invoices/'
          overwrite: false
          counter_separator: '_'
```

1.4.3 Rename

class Rename (*dest*[, *overwrite=False*][, *counter_separator=' '*])

Renames a file.

Parameters

- **name** (*str*) – The new filename. Can be a format string which uses file attributes from a filter.
- **overwrite** (*bool*) – specifies whether existing files should be overwritten. Otherwise it will start enumerating files (append a counter to the filename) to resolve naming conflicts. [Default: False]
- **counter_separator** (*str*) – specifies the separator between filename and the appended counter. Only relevant if **overwrite** is disabled. [Default: `'    '`]

Examples:

- Convert all .PDF file extensions to lowercase (.pdf):

Listing 33: config.yaml

```yaml
rules:
  - folders: '~/Desktop'
    filters:
      - extension: PDF
    actions:
      - rename: "{path.stem}.pdf"
```

- Convert **all** file extensions to lowercase:

Listing 34: config.yaml

```yaml
rules:
  - folders: '~/Desktop'
    filters:
      - Extension
    actions:
      - rename: "{path.stem}.{extension.lower}"
```

1.4.4 Trash

class Trash

Move a file into the trash.

Example:

- Move all JPGs and PNGs on the desktop which are older than one year into the trash:

Listing 35: config.yaml

```yaml
rules:
  - folders: '~/Desktop'
  - filters:
      - lastmodified:
          - days: 365
      - extension:
```

(continues on next page)

(continued from previous page)

```
          - png
          - jpg
  - actions:
      - trash
```

1.4.5 Shell

class Shell(*cmd: str*)
Executes a shell command

Parameters **cmd** (*str*) – The command to execute.

Example:

- (macOS) Open all pdfs on your desktop:

Listing 36: config.yaml

```
rules:
  - folders: '~/Desktop'
    filters:
      - extension: pdf
    actions:
      - shell: 'open "{path}"'
```

1.4.6 Python

class Python(*code*)
Execute python code in your config file.

Parameters **code** (*str*) – The python code to execute

Examples:

- A basic example that shows how to get the current file path and do some printing in a for loop. The `|` is yaml syntax for defining a string literal spanning multiple lines.

Listing 37: config.yaml

```
rules:
- folders: '~/Desktop'
  actions:
    - python: |
        print('The path of the current file is %s' % path)
        for _ in range(5):
            print('Heyho, its me from the loop')
```

- You can access filter data:

Listing 38: config.yaml

```
rules:
- folders: ~/Desktop
  filters:
```

(continues on next page)

(continued from previous page)

```
    - regex: '^(?P<name>.*)\.(?P<extension>.*)$'
  actions:
    - python: |
        print('Name: %s' % regex.name)
        print('Extension: %s' % regex.extension)
```

- You have access to all the python magic – do a google search for each filename starting with an underscore:

Listing 39: config.yaml

```
rules:
- folders: ~/Desktop
  filters:
    - filename:
        startswith: '_'
  actions:
    - python: |
        import webbrowser
        webbrowser.open('https://www.google.com/search?q=%s' % path.stem)
```

1.4.7 Echo

class Echo(*msg*)

Prints the given (formatted) message. This can be useful to test your rules, especially if you use formatted messages.

Parameters **msg** (*str*) – The message to print (can be formatted)

Example:

- Prints "Found old file" for each file older than one year:

Listing 40: config.yaml

```
rules:
  - folders: ~/Desktop
    filters:
      - lastmodified:
          days: 365
    actions:
      - echo: 'Found old file'
```

- Prints "Hello World!" and filepath for each file on the desktop:

Listing 41: config.yaml

```
rules:
  - folders:
      - ~/Desktop
    actions:
      - echo: 'Hello World! {path}'
```

- This will print something like `Found a PNG: "test.png"` for each file on your desktop:

Listing 42: config.yaml

```yaml
rules:
  - folders:
      - ~/Desktop
    filters:
      - Extension
    actions:
      - echo: 'Found a {extension.upper}: "{path.name}"'
```

- Show the `{basedir}` and `{path}` of all files in '~/Downloads', '~/Desktop' and their subfolders:

Listing 43: config.yaml

```yaml
rules:
  - folders:
      - ~/Desktop
      - ~/Downloads
    subfolders: true
    actions:
      - echo: 'Basedir: {basedir}'
      - echo: 'Path:    {path}'
```

If you find any bugs or have an idea for a new feature please don't hesitate to open an issue on GitHub.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index