

---

# **organize Documentation**

***Release 1.10.1***

**Thomas Feldmann**

**Jan 24, 2022**



---

## Contents

---

<b>1</b>	<b>Contents:</b>	<b>3</b>
1.1	Quickstart . . . . .	3
1.2	Configuration . . . . .	4
1.3	Filters . . . . .	10
1.4	Actions . . . . .	22
<b>2</b>	<b>Indices and tables</b>	<b>31</b>
	<b>Python Module Index</b>	<b>33</b>
	<b>Index</b>	<b>35</b>



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

<http://github.com/tfeldmann/organize>



## 1.1 Quickstart

### 1.1.1 Installation

Requirements: Python 3.6+

*organize* is installed via pip:

```
$ pip install organize-tool
```

If you want all the text extraction capabilities, install with *textract* like this:

```
$ sudo pip3 -U organize-tool[textract]
```

### 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/
```

(continues on next page)

(continued from previous page)

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

```
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

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

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

---

### Note:

- You can use environment variables in your folder names. On windows this means you can use %public%/Desktop, %APPDATA%, %PROGRAMDATA% etc.
- 

## 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

```
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

```
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

```
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

```
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`
- excludes 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

```
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 you 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

```
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

```
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

```
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

```
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 Created

**class Created** (*years=0, months=0, weeks=0, days=0, hours=0, minutes=0, seconds=0, mode='older',  
                  timezone=Timezone('Etc/UTC')*)

Matches files by created date

#### Parameters

- **years** (*int*) – specify number of years
- **months** (*int*) – specify number of months
- **weeks** (*float*) – specify number of weeks
- **days** (*float*) – specify number of days
- **hours** (*float*) – specify number of hours

- **minutes** (*float*) – specify number of minutes
- **seconds** (*float*) – specify number of seconds
- **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’)
- **timezone** (*str*) – specify timezone

#### 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 13: config.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 14: config.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 15: config.yaml

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

- Use specific timezone when processing files

Listing 16: config.yaml

```
rules:
- folders: '~/Documents'
  filters:
    - extension: pdf
    - created:
        timezone: "Europe/Moscow"
  actions:
    - move: '~/Documents/PDF/{created.day}/{created.hour}/'
```

### 1.3.2 Duplicate

#### **class Duplicate**

Finds duplicate files.

This filter compares files byte by byte and finds identical files with potentially different filenames.

#### **Returns**

- {duplicate} – path to the duplicate source

#### **Examples:**

- Show all duplicate files in your desktop and download folder (and their subfolders).

Listing 17: config.yaml

```
rules:
- folders:
    - ~/Desktop
    - ~/Downloads
  subfolders: true
  filters:
    - duplicate
  actions:
    - echo: "{path} is a duplicate of {duplicate}"
```

### 1.3.3 Exif

#### **class Exif** (\*required\_tags, \*\*tag\_filters)

Filter by image EXIF data

The *exif* filter can be used as a filter as well as a way to get exif information into your actions.

#### **Returns**

{exif} – a dict of all the collected exif information available in the file. Typically it consists of the following tags (if present in the file):

- {exif.image} – information related to the main image
- {exif.exif} – Exif information
- {exif.gps} – GPS information
- {exif.interoperability} – Interoperability information



**Examples:**

- Show available EXIF data of your pictures:

Listing 18: config.yaml

```
rules:
- folders: ~/Pictures
  subfolders: true
  filters:
  - exif
  actions:
  - echo: "{exif}"
```

- Copy all images which contain GPS information while keeping subfolder structure:

Listing 19: config.yaml

```
rules:
- folders: ~/Pictures
  subfolders: true
  filters:
  - exif:
      gps.gpsdate
  actions:
  - copy: ~/Pictures/with_gps/{relative_path}/
```

- Filter by camera manufacturer:

Listing 20: config.yaml

```
rules:
- folders: ~/Pictures
  subfolders: true
  filters:
  - exif:
      image.model: Nikon D3200
  actions:
  - move: '~/Pictures/My old Nikon/'
```

- Sort images by camera manufacturer. This will create folders for each camera model (for example “Nikon D3200”, “iPhone 6s”, “iPhone 5s”, “DMC-GX80”) and move the pictures accordingly:

Listing 21: config.yaml

```
rules:
- folders: ~/Pictures
  subfolders: true
  filters:
    - extension: jpg
    - exif:
        image.model
  actions:
    - move: '~/Pictures/{exif.image.model}/'
```

### 1.3.4 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 22: config.yaml

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

- Match multiple file extensions:

Listing 23: config.yaml

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

- Make all file extensions lowercase:

Listing 24: config.yaml

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

- Using extension lists:

Listing 25: config.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}'
```

### 1.3.5 FileContent

**class FileContent** (*expr*)

Matches file content 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**

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

Examples:

- Show the content of all your PDF files:
- Match an invoice with a regular expression and sort by customer:

Listing 26: config.yaml

```
rules:
  - folders: '~/Desktop'
    filters:
      - filecontent: 'Invoice.*Customer (?P<customer>\w+)'
```

(continues on next page)

(continued from previous page)

```
actions:
  - move: '~/Documents/Invoices/{filecontent.customer}/'
```

### 1.3.6 Filename

**class Filename** (*match*='\*', *startswith*="", *contains*="", *endswith*="", *case\_sensitive*=True)

Match files by filename

#### Parameters

- **match** (*str*) – A matching string in *simplematch*-syntax (<https://github.com/tfeldmann/simplematch>)
- **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 27: config.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 28: config.yaml

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

- Match all files starting with 'A' or 'B' containing '5' or '6' and ending with '\_end'

Listing 29: config.yaml

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

(continues on next page)

(continued from previous page)

```

- filename:
  startswith:
    - A
    - B
  contains:
    - 5
    - 6
  endswith: _end
  case_sensitive: false
actions:
- echo: 'Found a match.'
```

### 1.3.7 FileSize

**class FileSize** (\*conditions)

Matches files by file size

**Parameters** conditions (str) –

Accepts file size conditions, e.g: '>= 500 MB', '< 20k', '>0', '= 10 KiB'.

It is possible to define both lower and upper conditions like this: '>20k, < 1 TB', '>= 20 Mb, <25 Mb'. The filter will match if all given conditions are satisfied.

- Accepts all units from KB to YB.
- If no unit is given, kilobytes are assumed.
- If binary prefix is given (KiB, GiB) the size is calculated using base 1024.

**Returns**

- {filesize.bytes} – File size in bytes

**Examples:**

- Trash big downloads:

Listing 30: config.yaml

```

rules:
- folders: '~/Downloads'
  filters:
    - filesize: '> 0.5 GB'
  actions:
    - trash
```

- Move all JPEGs bigger > 1MB and <10 MB. Search all subfolders and keep the original relative path.

Listing 31: config.yaml

```

rules:
- folders: '~/Pictures'
  subfolders: true
  filters:
    - extension:
```

(continues on next page)

(continued from previous page)

```
- jpg
- jpeg
- filesize: '>1mb, <10mb'
actions:
- move: '~/Pictures/sorted/{relative_path}/'
```

### 1.3.8 LastModified

**class LastModified**(*years=0, months=0, weeks=0, days=0, hours=0, minutes=0, seconds=0, mode='older', timezone=Timezone('Etc/UTC')*)

Matches files by last modified date

#### Parameters

- **years** (*int*) – specify number of years
- **months** (*int*) – specify number of months
- **weeks** (*float*) – specify number of weeks
- **days** (*float*) – specify number of days
- **hours** (*float*) – specify number of hours
- **minutes** (*float*) – specify number of minutes
- **seconds** (*float*) – specify number of seconds
- **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')
- **timezone** (*str*) – specify timezone

#### 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 32: config.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 33: config.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 34: config.yaml

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

- Use specific timezone when processing files

Listing 35: config.yaml

```
rules:
- folders: '~/Documents'
  filters:
  - extension: pdf
  - lastmodified:
    timezone: "Europe/Moscow"
  actions:
  - move: '~/Documents/PDF/{lastmodified.day}/{lastmodified.hour}/'
```

### 1.3.9 MimeType

**class MimeType** (\*mimetypes)

Filter by MIME type associated with the file extension.

Supports a single string or list of MIME type strings as argument. The types don't need to be fully specified, for example "audio" matches everything from "audio/midi" to "audio/quicktime".

You can see a list of known MIME types on your system by running this oneliner:

```
python3 -c "import mimetypes as m; print('\n'.join(sorted(set(m.common_types.
↪values()) | set(m.types_map.values()))))"
```

**Examples:**

- Show MIME types:

Listing 36: config.yaml

```
rules:
  - folders: '~/Downloads'
    filters:
      - mimetype
    actions:
      - echo: '{mimetype}'
```

- Filter by “image” mimetype:

Listing 37: config.yaml

```
rules:
  - folders: '~/Downloads'
    filters:
      - mimetype: image
    actions:
      - echo: This file is an image: {mimetype}
```

- Filter by specific MIME type:

Listing 38: config.yaml

```
rules:
  - folders: '~/Desktop'
    filters:
      - mimetype: application/pdf
    actions:
      - echo: 'Found a PDF file'
```

- Filter by multiple specific MIME types:

Listing 39: config.yaml

```
rules:
  - folders: '~/Music'
    filters:
      - mimetype:
        - application/pdf
        - audio/midi
    actions:
      - echo: 'Found Midi or PDF.'
```

## 1.3.10 Python

**class** `Python` (*code*)

Use python code to filter files.

**Parameters** `code` (*str*) – The python code to execute. The code must contain a `return` statement.

**Returns**

- If your code returns `False` or `None` the file is filtered out, otherwise the file is passed on to the next filters.



- `{python}` contains the returned value. If you return a dictionary (for example `return {"some_key": some_value, "nested": {"k": 2}}`) it will be accessible via dot syntax in your actions: `{python.some_key}`, `{python.nested.k}`.

**Examples:**

- A file name reverser.

Listing 40: config.yaml

```
rules:
- folders: ~/Documents
  filters:
  - extension
  - python: |
      return {"reversed_name": path.stem[::-1]}
  actions:
  - rename: '{python.reversed_name}.{extension}'
```

- A filter for odd student numbers. Assuming the folder `~/Students` contains the files `student-01.jpg`, `student-01.txt`, `student-02.txt` and `student-03.txt` this rule will print "Odd student numbers: `student-01.txt`" and "Odd student numbers: `student-03.txt`"

Listing 41: config.yaml

```
rules:
- folders: ~/Students/
  filters:
  - python: |
      return int(path.stem.split('-')[1]) % 2 == 1
  actions:
  - echo: 'Odd student numbers: {path.name}'
```

- Advanced usecase. You can access data from previous filters in your python code. This can be used to match files and capturing names with a regular expression and then renaming the files with the output of your python script.

Listing 42: config.yaml

```
rules:
- folders: files
  filters:
  - extension: txt
  - regex: (?P<firstname>\w+)-(?P<lastname>\w+)\.*
  - python: |
      emails = {
          "Betts": "dbetts@mail.de",
          "Cornish": "acornish@google.com",
          "Bean": "dbean@aol.com",
          "Frey": "l-frey@frey.org",
      }
      if regex.lastname in emails: # get emails from wherever
          return {"mail": emails[regex.lastname]}
  actions:
  - rename: '{python.mail}.txt'
```

**Result:**

- Devonte-Betts.txt becomes dbetts@mail.de.txt
- Alaina-Cornish.txt becomes acornish@google.com.txt
- Dimitri-Bean.txt becomes dbean@aol.com.txt
- Lowri-Frey.txt becomes l-frey@frey.org.txt
- Someunknown-User.txt remains unchanged because the email is not found

## 1.3.11 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 43: config.yaml

```
rules:
- folders: '~/Desktop'
  filters:
  - regex: '^RG(\d{12})-sig\.pdf$'
  actions:
  - move: '~/Documents/Invoices/lund1/'
```

- 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 44: config.yaml

```
rules:
- folders: ~/Desktop
  filters:
  - regex: '^RG(?P<the_number>\d{12})-sig\.pdf$'
  actions:
  - move: ~/Documents/Invoices/lund1/{regex.the_number}.pdf
```

## 1.4 Actions

### 1.4.1 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 45: config.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 46: config.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 47: config.yaml

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

## 1.4.2 Delete

### **class Delete**

Delete a file from disk.

Deleted files have no recovery option! Using the *Trash* action is strongly advised for most use-cases!

#### **Example:**

- Delete all JPGs and PNGs on the desktop which are older than one year:

Listing 48: config.yaml

```
rules:
- folders: '~/Desktop'
- filters:
  - lastmodified:
    - days: 365
  - extension:
    - png
    - jpg
- actions:
  - delete
```

## 1.4.3 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 49: 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 50: 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 51: config.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 52: config.yaml

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

### 1.4.4 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 53: config.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 54: config.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 55: config.yaml

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

## 1.4.5 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 56: 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 57: config.yaml

```
rules:
- folders: ~/Desktop
  filters:
    - 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 58: config.yaml

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

## 1.4.6 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 59: config.yaml

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

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

Listing 60: config.yaml

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

## 1.4.7 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 61: config.yaml

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

## 1.4.8 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 62: config.yaml

```
rules:
- folders: '~/Desktop'
- filters:
    - lastmodified:
        - days: 365
    - extension:
        - png
        - jpg
- actions:
    - trash
```



### 1.4.9 macOS Tags

**class** `MacOSTags (*tags)`

Add macOS tags.

**Example:**

- Add a single tag:

Listing 63: config.yaml

```
rules:
- folders: '~/Documents/Invoices'
- filters:
  - filename:
    startswith: "Invoice"
  - extension: pdf
- actions:
  - macos_tags: Invoice
```

- Adding multiple tags (“Invoice” and “Important”):

Listing 64: config.yaml

```
rules:
- folders: '~/Documents/Invoices'
- filters:
  - filename:
    startswith: "Invoice"
  - extension: pdf
- actions:
  - macos_tags:
    - Important
    - Invoice
```

- Specify tag colors. Available colors are *none*, *gray*, *green*, *purple*, *blue*, *yellow*, *red*, *orange*.

Listing 65: config.yaml

```
rules:
- folders: '~/Documents/Invoices'
- filters:
  - filename:
    startswith: "Invoice"
  - extension: pdf
- actions:
  - macos_tags:
    - Important (green)
    - Invoice (purple)
```

- Add a templated tag with color:

Listing 66: config.yaml

```
rules:
- folders: '~/Documents/Invoices'
- filters:
  - created
```

(continues on next page)

(continued from previous page)

```
- actions:
  - macos_tags:
    - Year-{created.year} (red)
```

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`



### **a**

actions, [22](#)

### **f**

filters, [10](#)



## A

actions (*module*), 22

## C

Copy (*class in actions*), 22

Created (*class in filters*), 10

## D

Delete (*class in actions*), 24

Duplicate (*class in filters*), 12

## E

Echo (*class in actions*), 24

Exif (*class in filters*), 12

Extension (*class in filters*), 14

## F

FileContent (*class in filters*), 15

Filename (*class in filters*), 16

FileSize (*class in filters*), 17

filters (*module*), 10

## L

LastModified (*class in filters*), 18

## M

MacOSTags (*class in actions*), 29

MimeType (*class in filters*), 19

Move (*class in actions*), 25

## P

Python (*class in actions*), 26

Python (*class in filters*), 20

## R

Regex (*class in filters*), 22

Rename (*class in actions*), 27

## S

Shell (*class in actions*), 28

## T

Trash (*class in actions*), 28