urlwatch

Release 2.18

Mar 04, 2022

Contents

```
                _                 _        _         _____
 _   _ _ __| |_       _____ _| |_ ___| |__   |___ \
| | | | '__| \ \ /\ / / _` | __/ __| '_ \    __) |
| |_| | |  | |\ V  V / (_| | || (__| | | |  / __/
 \__,_|_|  |_| \_/\_/ \__,_|\__\___|_| |_| |_____|

                     ... monitors webpages for you
```

urlwatch is intended to help you watch changes in webpages and get notified (via e-mail, in your terminal or through various third party services) of any changes. The change notification will include the URL that has changed and a unified diff of what has changed.

1. Run `urlwatch` once to migrate your old data or start fresh
2. Use `urlwatch --edit` to customize your job list (this will create/edit `urls.yaml`)
3. Use `urlwatch --edit-config` if you want to set up e-mail sending
4. Add `urlwatch` to your crontab (`crontab -e`) to monitor webpages periodically

The checking interval is defined by how often you run `urlwatch`. You can use e.g. crontab.guru to figure out the schedule expression for the checking interval, we recommend not more often than 30 minutes (this would be `*/30 * * * *`). If you have never used cron before, check out the crontab command help.

On Windows, `cron` is not installed by default. Use the Windows Task Scheduler instead, or see this StackOverflow question for alternatives.

CHAPTER 1

The Handbook

1.1 Introduction

urlwatch monitors the output of webpages or arbitrary shell commands.

Every time you run *urlwatch*, it:

- retrieves the output and processes it
- compares it with the version retrieved the previous time ("diffing")
- if it finds any differences, generates a summary "report" that can be displayed or sent via one or more methods, such as email

1.1.1 Jobs

Each website or shell command to be monitored consitutes a "job".

The instructions for each such job are contained in a config file in the YAML format, accessible with the `urlwatch --edit` command. If you get an error, set your `$EDITOR` (or `$VISUAL`) environment variable in your shell with a command such as `export EDITOR=/bin/nano`.

Typically, the first entry ("key") in a job is a `name`, which can be anything you want and helps you identify what you're monitoring.

The second key is one of either `url`, `navigate` or `command`:

- `url` retrieves what is served by the web server,
- `navigate` handles more web pages requiring JavaScript to display the content to be monitored, and
- `command` runs a shell command.

You can then use optional keys to finely control various job's parameters.

Finally, you often use the `filter` key to select one or more filters to apply to the data after it is retrieved, to:

- select HTML: `css`, `xpath`, `element-by-class`, `element-by-id`, `element-by-style`, `element-by-tag`
- make HTML more readable: `html2text`, `beautify`
- make PDFs readable: `pdf2text`
- make JSON more readable: `format-json`
- make iCal more readable: `ical2text`
- make binary readable: `hexdump`
- just detect changes: `sha1sum`
- edit text: `grep`, `grepi`, `strip`, `sort`

These filters can be chained. As an example, after retrieving an HTML document by using the `url` key, you can extract a selection with the `xpath` filter, convert this to text with `html2text`, use `grep` to extract only lines matching a specific regular expression, and then `sort` them:

```
name: "Sample urlwatch job definition"
url: "https://example.dummy/"
https_proxy: "http://dummy.proxy/"
max_tries: 2
filter:
  - xpath: '//section[@role="main"]'
  - html2text:
      method: pyhtml2text
      unicode_snob: true
      body_width: 0
      inline_links: false
      ignore_links: true
      ignore_images: true
      pad_tables: false
      single_line_break: true
  - grep: "lines I care about"
  - sort:
---
```

If you have more than one job, per YAML specifications, you separate them with a line containing only `---`.

1.1.2 Reporters

urlwatch can be configured to do something with its report besides (or in addition to) the default of displaying it on the console, such as one or more of:

- `email` (using SMTP)
- email using `mailgun`
- `slack`
- `pushbullet`
- `telegram`
- `matrix`
- `pushover`
- `stdout`

- `xmpp`

Reporters are configured in a separate file, see *Configuration*.

1.2 Dependencies

Mandatory requirements are required to run urlwatch. Depending on what optional features you want to use, you might also need to install additional packages – however, those are not needed to run urlwatch.

1.2.1 Mandatory Packages

- Python 3.5 or newer
- PyYAML
- minidb
- requests
- keyring
- appdirs
- lxml
- cssselect

The dependencies can be installed with (add `--user` to install to `$HOME`):

```
python3 -m pip install pyyaml minidb requests keyring appdirs lxml cssselect
```

1.2.2 Optional Packages

Optional packages can be installed using:

```
python3 -m pip install <packagename>
```

Where `<packagename>` is one of the following:

Feature	Python package(s) to install
Pushover reporter	chump
Pushbullet reporter	pushbullet.py
Matrix reporter	matrix_client and markdown2
stdout reporter with color on Windows	colorama
browser job kind	requests-html
Unit testing	pycodestyle, docutils, Pygments and dependencies for other features as needed
beautify filter	beautifulsoup4; optional dependencies (for `<script>` and `<style>` tags): jsbeautifier and cssbeautifier
pdf2text filter	pdftotext and its OS-specific dependencies (see the above link)
ocr filter	pytesseract and Pillow and Tesseract OCR)
XMPP reporter	aioxmpp

1.3 Jobs

Jobs are the kind of things that *urlwatch* can monitor.

The list of jobs to run are contained in the configuration file `urls.yaml`, accessed with the command `urlwatch --edit`, each separated by a line containing only `---`.

While optional, it is recommended that each job starts with a `name` entry:

```
name: "This is a human-readable name/label of the job"
```

1.3.1 URL

This is the main job type – it retrieves a document from a web server:

```
name: "urlwatch homepage"
url: "https://thp.io/2008/urlwatch/"
```

Required keys:

- `url`: The URL to the document to watch for changes

Job-specific optional keys:

- `cookies`: Cookies to send with the request (see *Advanced Topics*)
- `method`: HTTP method to use (default: `GET`)
- `data`: HTTP POST/PUT data
- `ssl_no_verify`: Do not verify SSL certificates (true/false)
- `ignore_cached`: Do not use cache control (ETag/Last-Modified) values (true/false)
- `http_proxy`: Proxy server to use for HTTP requests
- `https_proxy`: Proxy server to use for HTTPS requests
- `headers`: HTTP header to send along with the request
- `encoding`: Override the character encoding from the server (see *Advanced Topics*)
- `timeout`: Override the default socket timeout (see *Advanced Topics*)
- `ignore_connection_errors`: Ignore (temporary) connection errors (see *Advanced Topics*)
- `ignore_http_error_codes`: List of HTTP errors to ignore (see *Advanced Topics*)
- `ignore_timeout_errors`: Do not report errors when the timeout is hit
- `ignore_too_many_redirects`: Ignore redirect loops (see *Advanced Topics*)

(Note: `url` implies `kind: url`)

1.3.2 Navigate

This job type is a resource-intensive variant of "URL" to handle web pages requiring JavaScript in order to render the content to be monitored.

The optional `requests-html` package must be installed to run "Navigate" jobs (see *Dependencies*).

```
name: "A page with JavaScript"
navigate: "https://example.org/"
```

Required keys:

- `navigate`: URL to navigate to with the browser

Job-specific optional keys:

- none

As this job uses Requests-HTML to render the page in a headless Chromium instance, it requires massively more resources than a "URL" job. Use it only on pages where `url` does not give the right results.

Hint: in many instances instead of using "Navigate" you can monitor the output of an API called by the site during page loading containing the information you're after using the much faster "URL" job type.

(Note: `navigate` implies `kind: browser`)

1.3.3 Command

This job type allows you to watch the output of arbitrary shell commands, which is useful for e.g. monitoring a FTP uploader folder, output of scripts that query external devices (RPi GPIO), etc. . .

```
name: "What is in my Home Directory?"
command: "ls -al ~"
```

Required keys:

- `command`: The shell command to execute

Job-specific optional keys:

- none

(Note: `command` implies `kind: shell`)

1.3.4 Optional keys for all job types

- `name`: Human-readable name/label of the job
- `filter`: filters (if any) to apply to the output (can be tested with `--test-filter`)
- `max_tries`: Number of times to retry fetching the resource
- `diff_tool`: Command to a custom tool for generating diff text
- `diff_filter`: filters (if any) to apply to the diff result (can be tested with `--test-diff-filter`)
- `compared_versions`: Number of versions to compare for similarity
- `kind` (redundant): Either `url`, `shell` or `browser`. Automatically derived from the unique key (`url`, `command` or `navigate`) of the job type

1.3.5 Settings keys for all jobs at once

See *Job Defaults* for how to configure keys for all jobs at once.

1.4 Filters

Filters are currently used in two stages of processing:

- Applied to the downloaded page before diffing the changes (`filter`)
- Applied to the diff result before reporting the changes (`diff_filter`)

While creating your filter pipeline, you might want to preview what the filtered output looks like. You can do so by first configuring your job and then running urlwatch with the `--test-filter` command, passing in the index (from `--list`) or the URL/location of the job to be tested:

```
urlwatch --test-filter 1   # Test the first job in the list
urlwatch --test-filter https://example.net/   # Test the job with the given URL
```

The output of this command will be the filtered plaintext of the job, this is the output that will (in a real urlwatch run) be the input to the diff algorithm.

The `filter` is only applied to new content, the old content was already filtered when it was retrieved. This means that changes to `filter` are not visible when reporting unchanged contents (see *Display* for details), and the diff output will be between (old content with filter at the time old content was retrieved) and (new content with current filter).

Once urlwatch has collected at least 2 historic snapshots of a job (two different states of a webpage) you can use the command-line option `--test-diff-filter` to test your `diff_filter` settings; this will use historic data cached locally.

1.4.1 Built-in filters

The list of built-in filters can be retrieved using:

```
urlwatch --features
```

At the moment, the following filters are built-in:

- **beautify**: Beautify HTML
- **css**: Filter XML/HTML using CSS selectors
- **element-by-class**: Get all HTML elements by class
- **element-by-id**: Get an HTML element by its ID
- **element-by-style**: Get all HTML elements by style
- **element-by-tag**: Get an HTML element by its tag
- **format-json**: Convert to formatted json
- **grep**: Filter only lines matching a regular expression
- **grepi**: Remove lines matching a regular expression
- **hexdump**: Convert binary data to hex dump format
- **html2text**: Convert HTML to plaintext
- **pdf2text**: Convert PDF to plaintext
- **ical2text**: Convert iCalendar to plaintext
- **ocr**: Convert text in images to plaintext using Tesseract OCR

- **re.sub**: Replace text with regular expressions using Python's re.sub
- **reverse**: Reverse input items
- **sha1sum**: Calculate the SHA-1 checksum of the content
- **shellpipe**: Filter using a shell command
- **sort**: Sort input items
- **strip**: Strip leading and trailing whitespace
- **xpath**: Filter XML/HTML using XPath expressions

1.4.2 Picking out elements from a webpage

You can pick only a given HTML element with the built-in filter, for example to extract `<div id="something">...</div>` from a page, you can use the following in your `urls.yaml`:

```
url: http://example.org/idtest.html
filter:
  - element-by-id: something
```

Also, you can chain filters, so you can run html2text on the result:

```
url: http://example.net/id2text.html
filter:
  - element-by-id: something
  - html2text
```

1.4.3 Chaining multiple filters

The example urls.yaml file also demonstrates the use of built-in filters, here 3 filters are used: html2text, line-grep and whitespace removal to get just a certain info field from a webpage:

```
url: https://example.net/version.html
filter:
  - html2text
  - grep: "Current.*version"
  - strip
```

1.4.4 Extracting only the `<body>` tag of a page

If you want to extract only the body tag you can use this filter:

```
url: https://example.org/bodytag.html
filter:
  - element-by-tag: body
```

1.4.5 Filtering based on an XPath expression

To filter based on an XPath expression, you can use the `xpath` filter like so (see Microsoft's XPath Examples page for some other examples):

```
url: https://example.net/xpath.html
filter:
  - xpath: /html/body/marquee
```

This filters only the <marquee> elements directly below the <body> element, which in turn must be below the <html> element of the document, stripping out everything else.

1.4.6 Filtering based on CSS selectors

To filter based on a CSS selector, you can use the css filter like so:

```
url: https://example.net/css.html
filter:
  - css: ul#groceries > li.unchecked
```

This would filter only <li class="unchecked"> tags directly below <ul id="groceries"> elements.

Some limitations and extensions exist as explained in cssselect's documentation.

1.4.7 Using XPath and CSS filters with XML and exclusions

By default, XPath and CSS filters are set up for HTML documents. However, it is possible to use them for XML documents as well (these examples parse an RSS feed and filter only the titles and publication dates):

```
url: https://example.com/blog/xpath-index.rss
filter:
  - xpath:
      path: '//item/title/text()|//item/pubDate/text()'
      method: xml
```

```
url: http://example.com/blog/css-index.rss
filter:
  - css:
      selector: 'item > title, item > pubDate'
      method: xml
  - html2text: re
```

To match an element in an XML namespace, use a namespace prefix before the tag name. Use a : to seperate the namespace prefix and the tag name in an XPath expression, and use a | in a CSS selector.

```
url: https://example.net/feed/xpath-namespace.xml
filter:
  - xpath:
      path: '//item/media:keywords/text()'
      method: xml
      namespaces:
        media: http://search.yahoo.com/mrss/
```

```
url: http://example.org/feed/css-namespace.xml
filter:
  - css:
      selector: 'item > media|keywords'
      method: xml
      namespaces:
```

(continues on next page)

(continued from previous page)

```
        media: http://search.yahoo.com/mrss/
  - html2text
```

Alternatively, use the XPath expression `//*[name()='<tag_name>']` to bypass the namespace entirely.

Another useful option with XPath and CSS filters is `exclude`. Elements selected by this `exclude` expression are removed from the final result. For example, the following job will not have any `<a>` tag in its results:

```
url: https://example.org/css-exclude.html
filter:
  - css:
      selector: body
      exclude: a
```

1.4.8 Limiting the returned items from a CSS Selector or XPath

If you only want to return a subset of the items returned by a CSS selector or XPath filter, you can use two additional subfilters:

- `skip`: How many elements to skip from the beginning (default: 0)
- `maxitems`: How many elements to return at most (default: no limit)

For example, if the page has multiple elements, but you only want to select the second and third matching element (skip the first, and return at most two elements), you can use this filter:

```
url: https://example.net/css-skip-maxitems.html
filter:
  - css:
      selector: div.cpu
      skip: 1
      maxitems: 2
```

Dealing with duplicated results

If you get multiple results on one page, but you only expected one (e.g. because the page contains both a mobile and desktop version in the same HTML document, and shows/hides one via CSS depending on the viewport size), you can use `maxitems: 1` to only return the first item.

1.4.9 Filtering PDF documents

To monitor the text of a PDF file, you use the *pdf2text* filter. It requires the installation of the pdftotext library and any of its OS-specific dependencies.

This filter *must* be the first filter in a chain of filters, since it consumes binary data and outputs text data.

```
url: https://example.net/pdf-test.pdf
filter:
  - pdf2text
  - strip
```

If the PDF file is password protected, you can specify its password:

```
url: https://example.net/pdf-test-password.pdf
filter:
  - pdf2text:
      password: urlwatchsecret
  - strip
```

1.4.10 Sorting of webpage content

Sometimes a web page can have the same data between comparisons but it appears in random order. If that happens, you can choose to sort before the comparison.

```
url: https://example.net/sorting.txt
filter:
  - sort
```

The sort filter takes an optional `separator` parameter that defines the item separator (by default sorting is line-based), for example to sort text paragraphs (text separated by an empty line):

```
url: http://example.org/paragraphs.txt
filter:
  - sort:
      separator: "\n\n"
```

This can be combined with a boolean `reverse` option, which is useful for sorting and reversing with the same separator (using `%` as separator, this would turn `3%2%4%1` into `4%3%2%1`):

```
url: http://example.org/sort-reverse-percent.txt
filter:
  - sort:
      separator: '%'
      reverse: true
```

1.4.11 Reversing of lines or separated items

To reverse the order of items without sorting, the `reverse` filter can be used. By default it reverses lines:

```
url: http://example.com/reverse-lines.txt
filter:
  - reverse
```

This behavior can be changed by using an optional separator string argument (e.g. items separated by a pipe (`|`) symbol, as in `1|4|2|3`, which would be reversed to `3|2|4|1`):

```
url: http://example.net/reverse-separator.txt
filter:
  - reverse: '|'
```

Alternatively, the filter can be specified more verbose with a dict. In this example `"\n\n"` is used to separate paragraphs (items that are separated by an empty line):

```
url: http://example.org/reverse-paragraphs.txt
filter:
  - reverse:
      separator: "\n\n"
```

1.4.12 Watching Github releases

This is an example how to watch the GitHub "releases" page for a given project for the latest release version, to be notified of new releases:

```
url: https://github.com/thp/urlwatch/releases
filter:
  - xpath: '(//div[contains(@class,"release-timeline-tags")]//h4)[1]/a'
  - html2text: re
  - strip
```

1.4.13 Remove or replace text using regular expressions

Just like Python's `re.sub` function, there's the possibility to apply a regular expression and either remove of replace the matched text. The following example applies the filter 3 times:

1. Just specifying a string as the value will replace the matches with the empty string.
2. Simple patterns can be replaced with another string using "pattern" as the expression and "repl" as the replacement.
3. You can use groups (`()`) and back-reference them with `\1` (etc..) to put groups into the replacement string.

All features are described in Python's re.sub documentation (the `pattern` and `repl` values are passed to this function as-is, with the value of `repl` defaulting to the empty string).

```
url: https://example.com/regex-substitute.html
filter:
    - re.sub: '\s*href="[^"]*"'
    - re.sub:
        pattern: '<h1>'
        repl: 'HEADING 1: '
    - re.sub:
        pattern: '</([^>]*)>'
        repl: '<END OF TAG \1>'
```

If you want to enable certain flags (e.g. `re.MULTILINE`) in the call, this is possible by inserting an "inline flag" documented in flags in re.compile, here are some examples:

- `re.MULTILINE`: `(?m)` (Makes `^` match start-of-line and `$` match end-of-line)
- `re.DOTALL`: `(?s)` (Makes `.` also match a newline)
- `re.IGNORECASE`: `(?i)` (Perform case-insensitive matching)

This allows you, for example, to remove all leading spaces (only space character and tab):

```
url: http://example.com/leading-spaces.txt
filter:
  - re.sub: '(?m)^[ \t]*'
```

1.4.14 Using a shell script as a filter

While the built-in filters are powerful for processing markup such as HTML and XML, in some cases you might already know how you would filter your content using a shell command or shell script. The `shellpipe` filter allows you to start a shell and run custom commands to filter the content.

The text data to be filtered will be written to the standard input (`stdin`) of the shell process and the filter output will be taken from the shell's standard output (`stdout`).

For example, if you want to use `grep` tool with the case insensitive matching option (`-i`) and printing only the matching part of the line (`-o`), you can specify this as `shellpipe` filter:

```
url: https://example.net/shellpipe-grep.txt
filter:
  - shellpipe: "grep -i -o 'price: <span>.*</span>'"
```

This feature also allows you to use `sed`, `awk` and `perl` one-liners for text processing (of course, any text tool that works in a shell can be used). For example, this `awk` one-liner prepends the line number to each line:

```
url: https://example.net/shellpipe-awk-oneliner.txt
filter:
  - shellpipe: awk '{ print FNR " " $0 }'
```

You can also use a multi-line command for a more sophisticated shell script (`|` in YAML denotes the start of a text block):

```
url: https://example.org/shellpipe-multiline.txt
filter:
  - shellpipe: |
      FILENAME=`mktemp`
      # Copy the input to a temporary file, then pipe through awk
      tee $FILENAME | awk '/The numbers for (.*) are:/,/The next draw is on (.*)./'
      # Analyze the input file in some other way
      echo "Input lines: $(wc -l $FILENAME | awk '{ print $1 }')"
      rm -f $FILENAME
```

Within the `shellpipe` script, two environment variables will be set for further customization (this can be useful if you have a external shell script file that is used as filter for multiple jobs, but needs to treat each job in a slightly different way):

Environment variable	Contents
`$URLWATCH_JOB_NAME`	The name of the job (`name` key in jobs YAML)
`$URLWATCH_JOB_LOCATION`	The URL of the job, or command line (for shell jobs)

1.4.15 Converting text in images to plaintext

The `ocr` filter uses the Tesseract OCR engine to convert text in images to plain text. It requires two Python modules to be installed: pytesseract and Pillow. Any file formats supported by Pillow (PIL) are supported.

This filter *must* be the first filter in a chain of filters, since it consumes binary data and outputs text data.

```
url: https://example.net/ocr-test.png
filter:
  - ocr:
      timeout: 5
      language: eng
  - strip
```

The sub-filters `timeout` and `language` are optional:

- `timeout`: Timeout for the recognition, in seconds (default: 10 seconds)
- `language`: Text language (e.g. `fra` or `eng+fra`, default: `eng`)

1.5 Configuration

The global configuration for urlwatch contains basic settings for the generic behavior of urlwatch as well as the *Reporters*. You can edit it with:

```
urlwatch --edit-config
```

1.5.1 Display

In addition to always reporting changes (which is the whole point of urlwatch), urlwatch by default reports newly-added (`new`) pages and errors (`error`). You can change this behavior in the `display` section of the configuration:

```
display:
  new: true
  error: true
  unchanged: false
```

If you set `unchanged` to `true`, urlwatch will always report all pages that are checked but have not changed.

Filter changes are not applied for `unchanged`

Due to the way the filtered output is stored, `unchanged` will always report the old contents with the filters at the time of retrieval, meaning that any changes you do to the `filter` of a job will not be visible in the `unchanged` report. When the page changes, the new filter will be applied.

For this reason, `unchanged` cannot be used to test filters, you should use the `--test-filter` command line option to apply your current filter to the current page contents.

1.5.2 Reporters

Configuration of reporters is described in *Reporters*.

Here is an example configuration that reports on standard output in color, as well as HTML e-mail using `sendmail`:

```
report:
  text:
    details: true
    footer: true
    line_length: 75
  html:
    diff: unified
  email:
    enabled: true
    method: sendmail
    sendmail:
        path: /usr/sbin/sendmail
    from: 'urlwatch@example.org'
    to: 'you@example.org'
    html: true
    subject: '{count} changes: {jobs}'
  stdout:
    color: true
    enabled: true
```

Any reporter-specific configuration must be below the `report` key in the configuration.

Configuration settings like `text`, `html` and `markdown` will apply to all reporters that derive from that reporter (for example, the `stdout` reporter uses `text`, while the `email` reporter with `html: true` set uses `html`).

1.5.3 Job Defaults

If you want to change some settings for all your jobs, edit the `job_defaults` section in your config file:

```
job_defaults:
  all:
    diff_tool: wdiff
  url:
    ignore_connection_errors: true
```

The above config file sets all jobs to use `wdiff` as diff tool, and all `url` jobs to ignore connection errors.

The possible sub-keys to `job_defaults` are:

- `all`: Applies to all your jobs, independent of its kind
- `shell`: Applies only to `shell` jobs (with key `command`)
- `url`: Applies only to `url` jobs (with key `url`)
- `browser`: Applies only to `browser` jobs (with key `navigate`)

See *Jobs* about the different job kinds and what the possible keys are.

1.6 Reporters

By default *urlwatch* prints out information about changes to standard output, which is your terminal if you run it interactively. If running via *cron* or another scheduler service, it depends on how the scheduler is configured.

You can enable one or more additional reporters that are used to send change notifications. Please note that most reporters need additional dependencies installed.

See *Configuration* on how to edit the configuration.

1.6.1 Built-in reporters

The list of built-in reporters can be retrieved using:

```
urlwatch --features
```

At the moment, the following reporters are built-in:

- **stdout**: Print summary on stdout (the console)
- **email**: Send summary via e-mail / SMTP
- **mailgun**: Custom email reporter that uses Mailgun
- **matrix**: Custom Matrix reporter
- **pushbullet**: Send summary via pushbullet.com
- **pushover**: Send summary via pushover.net

- **slack**: Custom Slack reporter
- **telegram**: Custom Telegram reporter
- **ifttt**: Send summary via IFTTT

1.6.2 Pushover

You can configure urlwatch to send real time notifications about changes via Pushover. To enable this, ensure you have the `chump` python package installed (see *Dependencies*). Then edit your config (`urlwatch --edit-config`) and enable pushover. You will also need to add to the config your Pushover user key and a unique app key (generated by registering urlwatch as an application on your Pushover account.

You can send to a specific device by using the device name, as indicated when you add or view your list of devices in the Pushover console. For example `device: 'MyPhone'`, or `device: 'MyLaptop'`. To send to *all* of your devices, set `device: null` in your config (`urlwatch --edit-config`) or leave out the device configuration completely.

Setting the priority is possible via the `priority` config option, which can be `lowest`, `low`, `normal`, `high` or `emergency`. Any other setting (including leaving the option unset) maps to `normal`.

1.6.3 Pushbullet

Pushbullet notifications are configured similarly to Pushover (see above). You'll need to add to the config your Pushbullet Access Token, which you can generate at https://www.pushbullet.com/#settings

1.6.4 Telegram

Telegram notifications are configured using the Telegram Bot API. For this, you'll need a Bot API token and a chat id (see https://core.telegram.org/bots). Sample configuration:

```
telegram:
  bot_token: '999999999:3tOhy2CuZE0pTaCtszRfKpnagOG8IQbP5gf' # your bot api token
  chat_id: '88888888' # the chat id where the messages should be sent
  enabled: true
```

To set up Telegram, from your Telegram app, chat up BotFather (New Message, Search, "BotFather"), then say `/newbot` and follow the instructions. Eventually it will tell you the bot token (in the form seen above, `<number>:<random string>`) - add this to your config file.

You can then click on the link of your bot, which will send the message `/start`. At this point, you can use the command `urlwatch --telegram-chats` to list the private chats the bot is involved with. This is the chat ID that you need to put into the config file as `chat_id`. You may add multiple chat IDs as a YAML list:

```
telegram:
  bot_token: '999999999:3tOhy2CuZE0pTaCtszRfKpnagOG8IQbP5gf' # your bot api token
  chat_id:
    - '11111111'
    - '22222222'
  enabled: true
```

Don't forget to also enable the reporter.

1.6.5 Slack

Slack notifications are configured using "Slack Incoming Webhooks". Here is a sample configuration:

```
slack:
  webhook_url: 'https://hooks.slack.com/services/T50TXXXXU/BDVYYYYYY/
↪PWTqwyFM7CcCfGnNzdyDYZ'
  enabled: true
```

To set up Slack, from you Slack Team, create a new app and activate "Incoming Webhooks" on a channel, you'll get a webhook URL, copy it into the configuration as seen above.

You can use the command `urlwatch --test-slack` to test if the Slack integration works.

1.6.6 IFTTT

To configure IFTTT events, you need to retrieve your key from here:

https://ifttt.com/maker_webhooks/settings

The URL shown in "Account Info" has the following format:

```
https://maker.ifttt.com/use/{key}
```

In this URL, `{key}` is your API key. The configuration should look like this (you can pick any event name you want):

```
ifttt:
  enabled: true
  key: aA12abC3D456efgHIjkl7m
  event: event_name_you_want
```

The event will contain three values in the posted JSON:

- `value1`: The type of change (`new`, `changed`, `unchanged` or `error`)
- `value2`: The name of the job (`name` key in `jobs.yaml`)
- `value3`: The location of the job (`url`, `command` or `navigate` key in `jobs.yaml`)

These values will be passed on to the Action in your Recipe.

1.6.7 Matrix

You can have notifications sent to you through the Matrix protocol.

To achieve this, you first need to register a Matrix account for the bot on any homeserver.

You then need to acquire an access token and room ID, using the following instructions adapted from this guide:

1. Open Riot.im in a private browsing window
2. Register/Log in as your bot, using its user ID and password.
3. Set the display name and avatar, if desired.
4. In the settings page, select the "Help & About" tab, scroll down to the bottom and click Access Token: <click to reveal>.
5. Copy the highlighted text to your configuration.
6. Join the room that you wish to send notifications to.

7. Go to the Room Settings (gear icon) and copy the *Internal Room ID* from the bottom.
8. Close the private browsing window **but do not log out, as this invalidates the Access Token**.

Here is a sample configuration:

```
matrix:
  homeserver: https://matrix.org
  access_token: "YOUR_TOKEN_HERE"
  room_id: "!roomroomroom:matrix.org"
  enabled: true
```

You will probably want to use the following configuration for the `markdown` reporter, if you intend to post change notifications to a public Matrix room, as the messages quickly become noisy:

```
markdown:
  details: false
  footer: false
  minimal: true
  enabled: true
```

1.6.8 E-Mail via GMail SMTP

You need to configure your GMail account to allow for "less secure" (password-based) apps to login:

1. Go to https://myaccount.google.com/
2. Click on "Sign-in & security"
3. Scroll all the way down to "Allow less secure apps" and enable it

You do not want to do this with your primary GMail account, but rather on a separate account that you create just for sending mails via urlwatch. Allowing less secure apps and storing the password (even if it's in the keychain) is not good security practice for your primary account.

Now, start the configuration editor: `urlwatch --edit-config`

These are the keys you need to configure:

- `report/email/enabled`: `true`
- `report/email/from`: `your.username@gmail.com` (edit accordingly)
- `report/email/method`: `smtp`
- `report/email/smtp/host`: `smtp.gmail.com`
- `report/email/smtp/auth`: `true`
- `report/email/smtp/port`: `587`
- `report/email/smtp/starttls`: `true`
- `report/email/to`: The e-mail address you want to send reports to

Now, for setting the password, it's not stored in the config file, but in your keychain. To store the password, run: `urlwatch --smtp-login` and enter your password.

1.6.9 E-Mail via Amazon Simple E-Mail Service (SES)

Start the configuration editor: `urlwatch --edit-config`

These are the keys you need to configure:

- `report/email/enabled`: `true`
- `report/email/from`: `you@verified_domain.com` (edit accordingly)
- `report/email/method`: `smtp`
- `report/email/smtp/host`: `email-smtp.us-west-2.amazonaws.com` (edit accordingly)
- `report/email/smtp/user`: `ABCDEFGHIJ1234567890` (edit accordingly)
- `report/email/smtp/auth`: `true`
- `report/email/smtp/port`: `587` (25 or 465 also work)
- `report/email/smtp/starttls`: `true`
- `report/email/to`: The e-mail address you want to send reports to

The password is not stored in the config file, but in your keychain. To store the password, run: `urlwatch --smtp-login` and enter your password.

1.6.10 SMTP login without keyring

If for whatever reason you cannot use a keyring to store your password (for example, when using it from a `cron` job) you can also set the `insecure_password` option in the SMTP config:

- `report/email/smtp/auth`: `true`
- `report/email/smtp/insecure_password`: `secret123`

The `insecure_password` key will be preferred over the data stored in the keyring. Please note that as the name says, storing the password as plaintext in the configuration is insecure and bad practice, but for an e-mail account that's only dedicated for sending mails this might be a way. **Never ever use this with your your primary e-mail account!** Seriously! Create a throw-away GMail (or other) account just for sending out those e-mails or use local `sendmail` with a mail server configured instead of relying on SMTP and password auth.

Note that this makes it really easy for your password to be picked up by software running on your machine, by other users logged into the system and/or for the password to appear in log files accidentally.

1.6.11 XMPP

You can have notifications sent to you through the *XMPP protocol*.

To achieve this, you should register a new XMPP account that is just used for urlwatch.

Here is a sample configuration:

```
xmpp:
  enabled: true
  sender: "BOT_ACCOUNT_NAME"
  recipient: "YOUR_ACCOUNT_NAME"
```

The password is not stored in the config file, but in your keychain. To store the password, run: `urlwatch --xmpp-login` and enter your password.

If for whatever reason you cannot use a keyring to store your password you can also set the `insecure_password` option in the XMPP config. For more information about the security implications, see *SMTP login without keyring*.

1.7 Advanced Topics

1.7.1 Adding URLs from the command line

Quickly adding new URLs to the job list from the command line:

```
urlwatch --add url=http://example.org,name=Example
```

1.7.2 Using word-based differences

You can also specify an external `diff`-style tool (a tool that takes two filenames (old, new) as parameter and returns on its standard output the difference of the files), for example to use GNU `wdiff` to get word-based differences instead of line-based difference:

```
url: https://example.com/
diff_tool: wdiff
```

Note that `diff_tool` specifies an external command-line tool, so that tool must be installed separately (e.g. `apt install wdiff` on Debian or `brew install wdiff` on macOS). Coloring is supported for `wdiff`-style output, but potentially not for other diff tools.

1.7.3 Ignoring connection errors

In some cases, it might be useful to ignore (temporary) network errors to avoid notifications being sent. While there is a `display.error` config option (defaulting to `true`) to control reporting of errors globally, to ignore network errors for specific jobs only, you can use the `ignore_connection_errors` key in the job list configuration file:

```
url: https://example.com/
ignore_connection_errors: true
```

Similarly, you might want to ignore some (temporary) HTTP errors on the server side:

```
url: https://example.com/
ignore_http_error_codes: 408, 429, 500, 502, 503, 504
```

or ignore all HTTP errors if you like:

```
url: https://example.com/
ignore_http_error_codes: 4xx, 5xx
```

1.7.4 Overriding the content encoding

For web pages with misconfigured HTTP headers or rare encodings, it may be useful to explicitly specify an encoding from Python's Standard Encodings.

```
url: https://example.com/
encoding: utf-8
```

1.7.5 Changing the default timeout

By default, url jobs timeout after 60 seconds. If you want a different timeout period, use the `timeout` key to specify it in number of seconds, or set it to 0 to never timeout.

```
url: https://example.com/
timeout: 300
```

1.7.6 Supplying cookie data

It is possible to add cookies to HTTP requests for pages that need it, the YAML syntax for this is:

```
url: http://example.com/
cookies:
    Key: ValueForKey
    OtherKey: OtherValue
```

1.7.7 Comparing with several latest snapshots

If a webpage frequently changes between several known stable states, it may be desirable to have changes reported only if the webpage changes into a new unknown state. You can use `compared_versions` to do this.

```
url: https://example.com/
compared_versions: 3
```

In this example, changes are only reported if the webpage becomes different from the latest three distinct states. The differences are shown relative to the closest match.

1.7.8 Receiving a report every time urlwatch runs

If you are watching pages that change seldomly, but you still want to be notified daily if `urlwatch` still works, you can watch the output of the `date` command, for example:

```
name: "urlwatch watchdog"
command: "date"
```

Since the output of `date` changes every second, this job should produce a report every time urlwatch is run.

1.7.9 Using Redis as a cache backend

If you want to use Redis as a cache backend over the default SQLite3 file:

```
urlwatch --cache=redis://localhost:6379/
```

There is no migration path from the SQLite3 format, the cache will be empty the first time Redis is used.

1.7.10 Watching changes on .onion (Tor) pages

Since pages on the Tor Network are not accessible via public DNS and TCP, you need to either configure a Tor client as HTTP/HTTPS proxy or use the `torify(1)` tool from the `tor` package (`apt install tor` on Debian, `brew`

`install tor` on macOS). Setting up Tor is out of scope for this document. On a properly set up Tor installation, one can just prefix the `urlwatch` command with the `torify` wrapper to access .onion pages:

```
torify urlwatch
```

1.7.11 Watching Facebook Page Events

If you want to be notified of new events on a public Facebook page, you can use the following job pattern, replace `PAGE` with the name of the page (can be found by navigating to the events page on your browser):

```
url: http://m.facebook.com/PAGE/pages/permalink/?view_type=tab_events
filter:
  - css:
      selector: div#objects_container
      exclude: 'div.x, #m_more_friends_who_like_this, img'
  - re.sub:
      pattern: '(/events/\d*)[^"]*'
      repl: '\1'
  - html2text: pyhtml2text
```

1.7.12 Only show added or removed lines

The `diff_filter` feature can be used to filter the diff output text with the same tools (see filters) used for filtering web pages.

In order to show only diff lines with added lines, use:

```
url: http://example.com/things-get-added.html
diff_filter:
  - grep: '^[@+]'
```

This will only keep diff lines starting with `@` or `+`. Similarly, to only keep removed lines:

```
url: http://example.com/things-get-removed.html
diff_filter:
  - grep: '^[@-]'
```

More sophisticated diff filtering is possibly by combining existing filters, writing a new filter or using `shellpipe` to delegate the filtering/processing of the diff output to an external tool.

1.7.13 Pass diff output to a custom script

In some situations, it might be useful to run a script with the diff as input when changes were detected (e.g. to start an update or process something). This can be done by combining `diff_filter` with the `shellpipe` filter, which can be any custom script.

The output of the custom script will then be the diff result as reported by urlwatch, so if it outputs any status, the `CHANGED` notification that urlwatch does will contain the output of the custom script, not the original diff. This can even have a "normal" filter attached to only watch links (the `css: a` part of the filter definitions):

```
url: http://example.org/downloadlist.html
filter:
  - css: a
```

(continues on next page)

(continued from previous page)

```
diff_filter:
  - shellpipe: /usr/local/bin/process_new_links.sh
```

1.7.14 Setting the content width for `html2text` (`lynx` method)

When using the `lynx` method in the `html2text` filter, it uses a default width that will cause additional line breaks to be inserted.

To set the `lynx` output width to 400 characters, use this filter setup:

```
url: http://example.com/longlines.html
filter:
  - html2text:
      method: lynx
      width: 400
```

1.8 Deprecated Features

This page lists the features that are deprecated and steps to update your configuration to use the replacements (if any).

1.8.1 string-based filter definitions (since 2.19)

With urlwatch 2.19, string-based filter lists are deprecated, because they are not as flexible as dict-based filter lists and had some problems (e.g. `:` and `,` are treated in a special way and cannot be used in subfilters easily). If you have a filter definition like this:

```
filter: css:body,html2text:re,strip
```

You can get the same results with a filter definition like this:

```
filter:
  - css:
      selector: body
  - html2text:
      method: re
  - strip
```

Since `selector` is the default subfilter for `css`, and `method` is the default subfilter for `html2text`, this can also be written as:

```
filter:
  - css: body
  - html2text: re
  - strip
```

If you just have a single filter such as:

```
filter: html2text
```

You can change this filter to dict-based using:

```
filter:
  - html2text
```

1.8.2 keyring setting in SMTP reporter configuration (since 2.18)

Since version 2.18, the SMTP reporter configuration now uses `auth` to decide if SMTP authentication should be done or not. Previously, this setting was called `keyring`. If you have an old configuration like this:

```
report:
  email:
    smtp:
      host: localhost
      keyring: false
      port: 25
      starttls: true
    subject: '{count} changes: {jobs}'
```

You can change the setting to this (replace `keyring` with `auth`):

```
report:
  email:
    smtp:
      host: localhost
      auth: false
      port: 25
      starttls: true
    subject: '{count} changes: {jobs}'
```

1.9 Migration from 1.x

Migration from urlwatch 1.x should be automatic on first start. Here is a quick rundown of changes in 2.0:

- URLs are stored in a YAML file now, with direct support for specifying names for jobs, different job kinds, directly applying filters, selecting the HTTP request method, specifying POST data as dictionary and much more
- The cache directory has been replaced with a SQLite 3 database file "cache.db" in minidb format, storing all change history (use `--gc-cache` to remove old changes if you don't need them anymore) for further analysis
- The hooks mechanism has been replaced with support for creating new job kinds by subclassing, new filters (also by subclassing) as well as new reporters (pieces of code that put the results somewhere, for example the default installation contains the "stdout" reporter that writes to the console and the "email" reporter that can send HTML and text e-mails)
- A configuration file - urlwatch.yaml - has been added for specifying user preferences instead of having to supply everything via the command line

CHAPTER 2

Indices and tables

- genindex
- search