Paperless Documentation

Release 2.5.0

Daniel Quinn

Nov 03, 2018

Contents

Paperless is a simple Django application running in two parts: a *consumer* (the thing that does the indexing) and the *webserver* (the part that lets you search & download already-indexed documents). If you want to learn more about its functions keep on reading after the installation section.

CHAPTER 1

Why This Exists

Paper is a nightmare. Environmental issues aside, there's no excuse for it in the 21st century. It takes up space, collects dust, doesn't support any form of a search feature, indexing is tedious, it's heavy and prone to damage & loss.

I wrote this to make "going paperless" easier. I do not have to worry about finding stuff again. I feed documents right from the post box into the scanner and then shred them. Perhaps you might find it useful too.

CHAPTER 2

Contents

2.1 Requirements

You need a Linux machine or Unix-like setup (theoretically an Apple machine should work) that has the following software installed:

- Python3 (with development libraries, pip and virtualenv)
- GNU Privacy Guard
- Tesseract, plus its language files matching your document base.
- Imagemagick version 6.7.5 or higher
- unpaper
- libpoppler-cpp-dev PDF rendering library

Notably, you should confirm how you access your Python3 installation. Many Linux distributions will install Python3 in parallel to Python2, using the names `python3` and `python` respectively. The same goes for `pip3` and `pip`. Running Paperless with Python2 will likely break things, so make sure that you're using the right version.

For the purposes of simplicity, `python` and `pip` is used everywhere to refer to their Python3 versions.

In addition to the above, there are a number of Python requirements, all of which are listed in a file called `requirements.txt` in the project root directory.

If you're not working on a virtual environment (like Docker), you should probably be using a virtualenv, but that's your call. The reasons why you might choose a virtualenv or not aren't really within the scope of this document. Needless to say if you don't know what a virtualenv is, you should probably figure that out before continuing.

2.1.1 Problems with Imagemagick & PDFs

Some users have run into problems with getting ImageMagick to do its thing with PDFs. Often this is the case with Apple systems using HomeBrew, but other Linuxes have been a problem as well. The solution appears to be to install ghostscript as well as ImageMagick:

```
$ brew install ghostscript
$ brew install imagemagick
$ brew install libmagic
```

2.1.2 Python-specific Requirements: No Virtualenv

If you don't care to use a virtual env, then installation of the Python dependencies is easy:

```
$ pip install --user --requirement /path/to/paperless/requirements.txt
```

This will download and install all of the requirements into `${HOME}/.local`. Remember that your distribution may be using `pip3` as mentioned above.

2.1.3 Python-specific Requirements: Virtualenv

Using a virtualenv for this is pretty straightforward: create a virtualenv, enter it, and install the requirements using the `requirements.txt` file:

```
$ virtualenv --python=/path/to/python3 /path/to/arbitrary/directory
$ . /path/to/arbitrary/directory/bin/activate
$ pip install  --requirement /path/to/paperless/requirements.txt
```

Now you're ready to go. Just remember to enter (activate) your virtualenv whenever you want to use Paperless.

2.1.4 Documentation

As generation of the documentation is not required for the use of Paperless, dependencies for this process are not included in `requirements.txt`. If you'd like to generate your own docs locally, you'll need to:

```
$ pip install sphinx
```

and then cd into the `docs` directory and type `make html`.

If you are using Docker, you can use the following commands to build the documentation and run a webserver serving it on port 8001:

```
$ pwd
/path/to/paperless

$ docker build -t paperless:docs -f docs/Dockerfile .
$ docker run --rm -it -p "8001:8000" paperless:docs
```

2.2 Setup

Paperless isn't a very complicated app, but there are a few components, so some basic documentation is in order. If you follow along in this document and still have trouble, please open an issue on GitHub so I can fill in the gaps.

2.2.1 Download

The source is currently only available via GitHub, so grab it from there, either by using `git`:

```
$ git clone https://github.com/danielquinn/paperless.git
$ cd paperless
```

or just download the tarball and go that route:

```
$ cd to the directory where you want to run Paperless
$ wget https://github.com/danielquinn/paperless/archive/master.zip
$ unzip master.zip
$ cd paperless-master
```

2.2.2 Installation & Configuration

You can go multiple routes with setting up and running Paperless:

- The *bare metal route*
- The *docker route*

The *docker route* is quick & easy.

The *bare metal route* is a bit more complicated to setup but makes it easier should you want to contribute some code back.

Standard (Bare Metal)

1. Install the requirements as per the *requirements* page.
2. Within the extract of master.zip go to the `src` directory.
3. Copy `../paperless.conf.example` to `/etc/paperless.conf` and open it in your favourite editor. As this file contains passwords. It should only be readable by user root and paperless! Set the values for:

   Set the values for:

   - `PAPERLESS_CONSUMPTION_DIR`: this is where your documents will be dumped to be consumed by Paperless.
   - `PAPERLESS_OCR_THREADS`: this is the number of threads the OCR process will spawn to process document pages in parallel.
   - `PAPERLESS_PASSPHRASE`: this is only required if you want to use GPG to encrypt your document files. This is the passphrase Paperless uses to encrypt/decrypt the original documents. Don't worry about defining this if you don't want to use encryption (the default).
4. Initialise the SQLite database with `./manage.py migrate`.
5. Create a user for your Paperless instance with `./manage.py createsuperuser`. Follow the prompts to create your user.
6. Start the webserver with `./manage.py runserver <IP>:<PORT>`. If no specifc IP or port are given, the default is `127.0.0.1:8000` also known as http://localhost:8000/. You should now be able to visit your (empty) installation at Paperless webserver or whatever you chose before. You can login with the user/pass you created in #5.
7. In a separate window, change to the `src` directory in this repo again, but this time, you should start the consumer script with `./manage.py document_consumer`.

8. Scan something or put a file into the `CONSUMPTION_DIR`.
9. Wait a few minutes
10. Visit the document list on your webserver, and it should be there, indexed and downloadable.

Caution: This installation is not secure. Once everything is working head over to *Making things more permanent*

Docker Method

1. Install Docker.

   Caution: As mentioned earlier, this guide assumes that you use Docker natively under Linux. If you are using Docker Machine under Mac OS X or Windows, you will have to adapt IP addresses, volume-mounting, command execution and maybe more.

2. Install docker-compose.[1]

   Caution: If you want to use the included `docker-compose.yml.example` file, you need to have at least Docker version **1.10.0** and docker-compose version **1.6.0**.

   See the Docker installation guide on how to install the current version of Docker for your operating system or Linux distribution of choice. To get an up-to-date version of docker-compose, follow the docker-compose installation guide if your package repository doesn't include it.

3. Create a copy of `docker-compose.yml.example` as `docker-compose.yml` and a copy of `docker-compose.env.example` as `docker-compose.env`. You'll be editing both these files: taking a copy ensures that you can `git pull` to receive updates without risking merge conflicts with your modified versions of the configuration files.
4. Modify `docker-compose.yml` to your preferences, following the instructions in comments in the file. The only change that is a hard requirement is to specify where the consumption directory should mount.[#dockercomposeyml]_
5. Modify `docker-compose.env` and adapt the following environment variables:

   `PAPERLESS_PASSPHRASE` This is the passphrase Paperless uses to encrypt/decrypt the original document. If you aren't planning on using GPG encryption, you can just leave this undefined.

   `PAPERLESS_OCR_THREADS` This is the number of threads the OCR process will spawn to process document pages in parallel. If the variable is not set, Python determines the core-count of your CPU and uses that value.

   `PAPERLESS_OCR_LANGUAGES` If you want the OCR to recognize other languages in addition to the default English, set this parameter to a space separated list of three-letter language-codes after ISO 639-2/T. For a list of available languages – including their three letter codes – see the Alpine packagelist.

   `USERMAP_UID` and `USERMAP_GID` If you want to mount the consumption volume (directory `/consume` within the containers) to a host-directory – which you probably want to do – access rights might be an issue. The default user and group `paperless` in the containers have an id of 1000. The containers will enforce that the owning group of the consumption directory will be `paperless` to be able to delete

[1] You of course don't have to use docker-compose, but it simplifies deployment immensely. If you know your way around Docker, feel free to tinker around without using compose!

consumed documents. If your host-system has a group with an ID of 1000 and you don't want this group to have access rights to the consumption directory, you can use `USERMAP_GID` to change the id in the container and thus the one of the consumption directory. Furthermore, you can change the id of the default user as well using `USERMAP_UID`.

6. Run `docker-compose up -d`. This will create and start the necessary containers.
7. To be able to login, you will need a super user. To create it, execute the following command:

```
$ docker-compose run --rm webserver createsuperuser
```

This will prompt you to set a username (default `paperless`), an optional e-mail address and finally a password.

8. The default `docker-compose.yml` exports the webserver on your local port 8000. If you haven't adapted this, you should now be able to visit your Paperless webserver at `http://127.0.0.1:8000`. You can login with the user and password you just created.
9. Add files to consumption directory the way you prefer to. Following are two possible options:

(a) Mount the consumption directory to a local host path by modifying your `docker-compose.yml`:

```
diff --git a/docker-compose.yml b/docker-compose.yml
--- a/docker-compose.yml
+++ b/docker-compose.yml
@@ -17,9 +18,8 @@ services:
         volumes:
             - paperless-data:/usr/src/paperless/data
             - paperless-media:/usr/src/paperless/media
-            - /consume
+            - /local/path/you/choose:/consume
```

Danger: While the consumption container will ensure at startup that it can **delete** a consumed file from a host-mounted directory, it might not be able to **read** the document in the first place if the access rights to the file are incorrect.

Make sure that the documents you put into the consumption directory will either be readable by everyone (`chmod o+r file.pdf`) or readable by the default user or group id 1000 (or the one you have set with `USERMAP_UID` or `USERMAP_GID` respectively).

(b) Use `docker cp` to copy your files directly into the container:

```
$ # Identify your containers
$ docker-compose ps
        Name                       Command                State     Ports
-------------------------------------------------------------------------
paperless_consumer_1    /sbin/docker-entrypoint.sh ...   Exit 0
paperless_webserver_1   /sbin/docker-entrypoint.sh ...   Exit 0

$ docker cp /path/to/your/file.pdf paperless_consumer_1:/consume
```

`docker cp` is a one-shot-command, just like `cp`. This means that every time you want to consume a new document, you will have to execute `docker cp` again. You can of course automate this process, but option 1 is generally the preferred one.

Danger: `docker cp` will change the owning user and group of a copied file to the acting user at the destination, which will be `root`.

> You therefore need to ensure that the documents you want to copy into the container are readable by everyone (`chmod o+r file.pdf`) before copying them.

2.2.3 Making Things a Little more Permanent

Once you've tested things and are happy with the work flow, you should secure the installation and automate the process of starting the webserver and consumer.

Using a Real Webserver

The default is to use Django's development server, as that's easy and does the job well enough on a home network. However it is heavily discouraged to use it for more than that.

If you want to do things right you should use a real webserver capable of handling more than one thread. You will also have to let the webserver serve the static files (CSS, JavaScript) from the directory configured in `PAPERLESS_STATICDIR`. The default static files directory is `../static`.

For that you need to activate your virtual environment and collect the static files with the command:

```
$ cd <paperless directory>/src
$ ./manage.py collectstatic
```

Apache

This is a configuration supplied by steckerhalter on GitHub. It uses Apache and mod_wsgi, with a Paperless installation in `/home/paperless/`:

```
<VirtualHost *:80>
    ServerName example.com

    Alias /static/ /home/paperless/paperless/static/
    <Directory /home/paperless/paperless/static>
        Require all granted
    </Directory>

    WSGIScriptAlias / /home/paperless/paperless/src/paperless/wsgi.py
    WSGIDaemonProcess example.com user=paperless group=paperless threads=5 python-
↪path=/home/paperless/paperless/src:/home/paperless/.env/lib/python3.4/site-packages
    WSGIProcessGroup example.com

    <Directory /home/paperless/paperless/src/paperless>
        <Files wsgi.py>
            Require all granted
        </Files>
    </Directory>
</VirtualHost>
```

Nginx + Gunicorn

If you're using Nginx, the most common setup is to combine it with a Python-based server like Gunicorn so that Nginx is acting as a proxy. Below is a copy of a simple Nginx configuration fragment making use of a gunicorn instance

listening on localhost port 8000.

```
server {
    listen 80;

    index index.html index.htm index.php;
    access_log /var/log/nginx/paperless_access.log;
    error_log /var/log/nginx/paperless_error.log;

    location /static {

        autoindex on;
        alias <path-to-paperless-static-directory>;

    }

    location / {

        proxy_set_header Host $http_host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;

        proxy_pass http://127.0.0.1:8000;
    }
}
```

The gunicorn server can be started with the command:

```
$ <path-to-paperless-virtual-environment>/bin/gunicorn <path-to-paperless>/src/
↪paperless.wsgi -w 2
```

Standard (Bare Metal + Systemd)

If you're running on a bare metal system that's using Systemd, you can use the service unit files in the `scripts` directory to set this up.

1. You'll need to create a group and user called `paperless` (without login)
2. Setup Paperless to be in a place that this new user can read and write to.
3. Ensure `/etc/paperless` is readable by the `paperless` user.
4. Copy the service file from the `scripts` directory to `/etc/systemd/system`.

```
$ cp /path/to/paperless/scripts/paperless-consumer.service /etc/systemd/system/
$ cp /path/to/paperless/scripts/paperless-webserver.service /etc/systemd/system/
```

5. Edit the service file to point the `ExecStart` line to the proper location of your paperless install, referencing the appropriate Python binary. For example: `ExecStart=/path/to/python3 /path/to/paperless/src/manage.py document_consumer`.
6. Start and enable (so they start on boot) the services.

```
$ systemctl enable paperless-consumer
$ systemctl enable paperless-webserver
$ systemctl start paperless-consumer
$ systemctl start paperless-webserver
```

Standard (Bare Metal + Upstart)

Ubuntu 14.04 and earlier use the Upstart init system to start services during the boot process. To configure Upstart to run Paperless automatically after restarting your system:

1. Change to the directory where Upstart's configuration files are kept: `cd /etc/init`
2. Create a new file: `sudo nano paperless-server.conf`
3. In the newly-created file enter:

```
start on (local-filesystems and net-device-up IFACE=eth0)
stop on shutdown

respawn
respawn limit 10 5

script
  exec <path to paperless virtual environment>/bin/gunicorn <path to parperless>/
↪src/paperless.wsgi -w 2
end script
```

   Note that you'll need to replace `/srv/paperless/src/manage.py` with the path to the `manage.py` script in your installation directory.

   If you are using a network interface other than `eth0`, you will have to change `IFACE=eth0`. For example, if you are connected via WiFi, you will likely need to replace `eth0` above with `wlan0`. To see all interfaces, run `ifconfig -a`.

   Save the file.

4. Create a new file: `sudo nano paperless-consumer.conf`
5. In the newly-created file enter:

```
start on (local-filesystems and net-device-up IFACE=eth0)
stop on shutdown

respawn
respawn limit 10 5

script
  exec <path to paperless virtual environment>/bin/python <path to parperless>/
↪manage.py document_consumer
end script
```

   Replace the path placeholder and `eth0` with the appropriate value and save the file.

These two configuration files together will start both the Paperless webserver and document consumer processes when the file system and network interface specified is available after boot. Furthermore, if either process ever exits unexpectedly, Upstart will try to restart it a maximum of 10 times within a 5 second period.

Docker

If you're using Docker, you can set a restart-policy in the `docker-compose.yml` to have the containers automatically start with the Docker daemon.

2.3 Consumption

Once you've got Paperless setup, you need to start feeding documents into it. Currently, there are three options: the consumption directory, IMAP (email), and HTTP POST.

2.3.1 The Consumption Directory

The primary method of getting documents into your database is by putting them in the consumption directory. The `document_consumer` script runs in an infinite loop looking for new additions to this directory and when it finds them, it goes about the process of parsing them with the OCR, indexing what it finds, and encrypting the PDF (if `PAPERLESS_PASSPHRASE` is set), storing it in the media directory.

Getting stuff into this directory is up to you. If you're running Paperless on your local computer, you might just want to drag and drop files there, but if you're running this on a server and want your scanner to automatically push files to this directory, you'll need to setup some sort of service to accept the files from the scanner. Typically, you're looking at an FTP server like Proftpd or Samba.

So where is this consumption directory? It's wherever you define it. Look for the `CONSUMPTION_DIR` value in `settings.py`. Set that to somewhere appropriate for your use and put some documents in there. When you're ready, follow the *consumer* instructions to get it running.

Hooking into the Consumption Process

Sometimes you may want to do something arbitrary whenever a document is consumed. Rather than try to predict what you may want to do, Paperless lets you execute scripts of your own choosing just before or after a document is consumed using a couple simple hooks.

Just write a script, put it somewhere that Paperless can read & execute, and then put the path to that script in `paperless.conf` with the variable name of either `PAPERLESS_PRE_CONSUME_SCRIPT` or `PAPERLESS_POST_CONSUME_SCRIPT`. The script will be executed before or or after the document is consumed respectively.

Important: These scripts are executed in a **blocking** process, which means that if a script takes a long time to run, it can significantly slow down your document consumption flow. If you want things to run asynchronously, you'll have to fork the process in your script and exit.

What Can These Scripts Do?

It's your script, so you're only limited by your imagination and the laws of physics. However, the following values are passed to the scripts in order:

Pre-consumption script

- Document file name

A simple but common example for this would be creating a simple script like this:

`/usr/local/bin/ocr-pdf`

```
#!/usr/bin/env bash
pdf2pdfocr.py -i ${1}
```

/etc/paperless.conf

```
...
PAPERLESS_PRE_CONSUME_SCRIPT="/usr/local/bin/ocr-pdf"
...
```

This will pass the path to the document about to be consumed to /usr/local/bin/ocr-pdf, which will in turn call pdf2pdfocr.py on your document, which will then overwrite the file with an OCR'd version of the file and exit. At which point, the consumption process will begin with the newly modified file.

Post-consumption script

- Document id
- Generated file name
- Source path
- Thumbnail path
- Download URL
- Thumbnail URL
- Correspondent
- Tags

The script can be in any language you like, but for a simple shell script example, you can take a look at post-consumption-example.sh in the scripts directory in this project.

2.3.2 IMAP (Email)

Another handy way to get documents into your database is to email them to yourself. The typical use-case would be to be out for lunch and want to send a copy of the receipt back to your system at home. Paperless can be taught to pull emails down from an arbitrary account and dump them into the consumption directory where the process *above* will follow the usual pattern on consuming the document.

Some things you need to know about this feature:

- It's disabled by default. By setting the values below it will be enabled.
- It's been tested in a limited environment, so it may not work for you (please submit a pull request if you can!)
- It's designed to **delete mail from the server once consumed**. So don't go pointing this to your personal email account and wonder where all your stuff went.
- Currently, only one photo (attachment) per email will work.

So, with all that in mind, here's what you do to get it running:

1. Setup a new email account somewhere, or if you're feeling daring, create a folder in an existing email box and note the path to that folder.
2. In /etc/paperless.conf set all of the appropriate values in PATHS AND FOLDERS and SECURITY. If you decided to use a subfolder of an existing account, then make sure you set PAPERLESS_CONSUME_MAIL_INBOX accordingly here. You also have to set the PAPERLESS_EMAIL_SECRET to something you can remember 'cause you'll have to include that in every email you send.

3. Restart the *consumer*. The consumer will check the configured email account at startup and from then on every 10 minutes for something new and pulls down whatever it finds.
4. Send yourself an email! Note that the subject is treated as the file name, so if you set the subject to `Correspondent - Title - tag,tag,tag`, you'll get what you expect. Also, you must include the aforementioned secret string in every email so the fetcher knows that it's safe to import. Note that Paperless only allows the email title to consist of safe characters to be imported. These consist of alpha-numeric characters and `-_ ,.'`.
5. After a few minutes, the consumer will poll your mailbox, pull down the message, and place the attachment in the consumption directory with the appropriate name. A few minutes later, the consumer will import it like any other file.

2.3.3 HTTP POST

You can also submit a document via HTTP POST, so long as you do so after authenticating. To push your document to Paperless, send an HTTP POST to the server with the following name/value pairs:

- `correspondent`: The name of the document's correspondent. Note that there are restrictions on what characters you can use here. Specifically, alphanumeric characters, -, ,, ., and ' are ok, everything else is out. You also can't use the sequence ' - ' (space, dash, space).
- `title`: The title of the document. The rules for characters is the same here as the correspondent.
- `document`: The file you're uploading

Specify `enctype="multipart/form-data"`, and then POST your file with:

```
Content-Disposition: form-data; name="document"; filename="whatever.pdf"
```

An example of this in HTML is a typical form:

```
<form method="post" enctype="multipart/form-data">
    <input type="text" name="correspondent" value="My Correspondent" />
    <input type="text" name="title" value="My Title" />
    <input type="file" name="document" />
    <input type="submit" name="go" value="Do the thing" />
</form>
```

But a potentially more useful way to do this would be in Python. Here we use the requests library to handle basic authentication and to send the POST data to the URL.

```
import os

from hashlib import sha256

import requests
from requests.auth import HTTPBasicAuth

# You authenticate via BasicAuth or with a session id.
# We use BasicAuth here
username = "my-username"
password = "my-super-secret-password"

# Where you have Paperless installed and listening
url = "http://localhost:8000/push"

# Document metadata
```

(continues on next page)

(continued from previous page)

```
correspondent = "Test Correspondent"
title = "Test Title"

# The local file you want to push
path = "/path/to/some/directory/my-document.pdf"


with open(path, "rb") as f:

    response = requests.post(
        url=url,
        data={"title": title,  "correspondent": correspondent},
        files={"document": (os.path.basename(path), f, "application/pdf")},
        auth=HTTPBasicAuth(username, password),
        allow_redirects=False
    )

    if response.status_code == 202:

        # Everything worked out ok
        print("Upload successful")

    else:

        # If you don't get a 202, it's probably because your credentials
        # are wrong or something.  This will give you a rough idea of what
        # happened.

        print("We got HTTP status code: {}".format(response.status_code))
        for k, v in response.headers.items():
            print("{}: {}".format(k, v))
```

2.4 The REST API

Paperless makes use of the Django REST Framework standard API interface because of its inherent awesomeness. Conveniently, the system is also self-documenting, so to learn more about the access points, schema, what's accepted and what isn't, you need only visit `/api` on your local Paperless installation.

2.4.1 Uploading

File uploads in an API are hard and so far as I've been able to tell, there's no standard way of accepting them, so rather than crowbar file uploads into the REST API and endure that headache, I've left that process to a simple HTTP POST, documented on the *consumption page*.

2.5 Utilities

There's basically three utilities to Paperless: the webserver, consumer, and if needed, the exporter. They're all detailed here.

2.5.1 The Webserver

At the heart of it, Paperless is a simple Django webservice, and the entire interface is based on Django's standard admin interface. Once running, visiting the URL for your service delivers the admin, through which you can get a detailed listing of all available documents, search for specific files, and download whatever it is you're looking for.

How to Use It

The webserver is started via the `manage.py` script:

```
$ /path/to/paperless/src/manage.py runserver
```

By default, the server runs on localhost, port 8000, but you can change this with a few arguments, run `manage.py --help` for more information.

Add the option `--noreload` to reduce resource usage. Otherwise, the server continuously polls all source files for changes to auto-reload them.

Note that when exiting this command your webserver will disappear. If you want to run this full-time (which is kind of the point) you'll need to have it start in the background – something you'll need to figure out for your own system. To get you started though, there are Systemd service files in the `scripts` directory.

2.5.2 The Consumer

The consumer script runs in an infinite loop, constantly looking at a directory for documents to parse and index. The process is pretty straightforward:

1. Look in `CONSUMPTION_DIR` for a document. If one is found, go to #2. If not, wait 10 seconds and try again. On Linux, new documents are detected instantly via inotify, so there's no waiting involved.
2. Parse the document with Tesseract
3. Create a new record in the database with the OCR'd text
4. Attempt to automatically assign document attributes by doing some guesswork. Read up on the *guesswork documentation* for more information about this process.
5. Encrypt the document (if you have a passphrase set) and store it in the `media` directory under `documents/originals`.
6. Go to #1.

How to Use It

The consumer is started via the `manage.py` script:

```
$ /path/to/paperless/src/manage.py document_consumer
```

This starts the service that will consume documents as they appear in `CONSUMPTION_DIR`.

Note that this command runs continuously, so exiting it will mean your webserver disappears. If you want to run this full-time (which is kind of the point) you'll need to have it start in the background – something you'll need to figure out for your own system. To get you started though, there are Systemd service files in the `scripts` directory.

Some command line arguments are available to customize the behavior of the consumer. By default it will use `/etc/paperless.conf` values. Display the help with:

```
$ /path/to/paperless/src/manage.py document_consumer --help
```

2.5.3 The Exporter

Tired of fiddling with Paperless, or just want to do something stupid and are afraid of accidentally damaging your files? You can export all of your documents into neatly named, dated, and unencrypted files.

How to Use It

This too is done via the `manage.py` script:

```
$ /path/to/paperless/src/manage.py document_exporter /path/to/somewhere/
```

This will dump all of your unencrypted documents into `/path/to/somewhere` for you to do with as you please. The files are accompanied with a special file, `manifest.json` which can be used to *import the files* at a later date if you wish.

Docker

If you are *using Docker*, running the expoorter is almost as easy. To mount a volume for exports, follow the instructions in the `docker-compose.yml.example` file for the `/export` volume (making the changes in your own `docker-compose.yml` file, of course). Once you have the volume mounted, the command to run an export is:

```
$ docker-compose run --rm consumer document_exporter /export
```

If you prefer to use `docker run` directly, supplying the necessary commandline options:

```
$ # Identify your containers
$ docker-compose ps
        Name                       Command               State     Ports
-------------------------------------------------------------------------
paperless_consumer_1    /sbin/docker-entrypoint.sh ...   Exit 0
paperless_webserver_1   /sbin/docker-entrypoint.sh ...   Exit 0

$ # Make sure to replace your passphrase and remove or adapt the id mapping
$ docker run --rm \
    --volumes-from paperless_data_1 \
    --volume /path/to/arbitrary/place:/export \
    -e PAPERLESS_PASSPHRASE=YOUR_PASSPHRASE \
    -e USERMAP_UID=1000 -e USERMAP_GID=1000 \
    paperless document_exporter /export
```

2.5.4 The Importer

Looking to transfer Paperless data from one instance to another, or just want to restore from a backup? This is your go-to toy.

How to Use It

The importer works just like the exporter. You point it at a directory, and the script does the rest of the work:

```
$ /path/to/paperless/src/manage.py document_importer /path/to/somewhere/
```

Docker

Assuming that you've already gone through the steps above in the *export* section, then the easiest thing to do is just re-use the `/export` path you already setup:

```
$ docker-compose run --rm consumer document_importer /export
```

Similarly, if you're not using docker-compose, you can adjust the export instructions above to do the import.

2.5.5 The Re-tagger

Say you've imported a few hundred documents and now want to introduce a tag and apply its matching to all of the currently-imported docs. This problem is common enough that there's a tool for it.

How to Use It

This too is done via the `manage.py` script:

```
$ /path/to/paperless/src/manage.py document_retagger
```

That's it. It'll loop over all of the documents in your database and attempt to match all of your tags to them. If one matches, it'll be applied. And don't worry, you can run this as often as you like, it' won't double-tag a document.

2.6 Guesswork

During the consumption process, Paperless tries to guess some of the attributes of the document it's looking at. To do this it uses two approaches:

2.6.1 File Naming

Any document you put into the consumption directory will be consumed, but if you name the file right, it'll automatically set some values in the database for you. This is is the logic the consumer follows:

1. Try to find the correspondent, title, and tags in the file name following the pattern: `Date - Correspondent - Title - tag,tag,tag.pdf`. Note that the format of the date is **rigidly defined** as `YYYYMMDDHHMMSSZ` or `YYYYMMDDZ`. The `Z` refers "Zulu time" AKA "UTC". The tags are optional, so the format `Date - Correspondent - Title.pdf` works as well.
2. If that doesn't work, we skip the date and try this pattern: `Correspondent - Title - tag,tag,tag.pdf`.
3. If that doesn't work, we try to find the correspondent and title in the file name following the pattern: `Correspondent - Title.pdf`.
4. If that doesn't work, just assume that the name of the file is the title.

So given the above, the following examples would work as you'd expect:

- `20150314000700Z - Some Company Name - Invoice 2016-01-01 - money,invoices.pdf`

- `20150314Z - Some Company Name - Invoice 2016-01-01 - money,invoices.pdf`
- `Some Company Name - Invoice 2016-01-01 - money,invoices.pdf`
- `Another Company - Letter of Reference.jpg`
- `Dad's Recipe for Pancakes.png`

These however wouldn't work:

- `2015-03-14 00:07:00 UTC - Some Company Name, Invoice 2016-01-01, money, invoices.pdf`
- `2015-03-14 - Some Company Name, Invoice 2016-01-01, money, invoices.pdf`
- `Some Company Name, Invoice 2016-01-01, money, invoices.pdf`
- `Another Company- Letter of Reference.jpg`

2.6.2 Reading the Document Contents

After the consumer has tried to figure out what it could from the file name, it starts looking at the content of the document itself. It will compare the matching algorithms defined by every tag and correspondent already set in your database to see if they apply to the text in that document. In other words, if you defined a tag called `Home Utility` that had a `match` property of `bc hydro` and a `matching_algorithm` of `literal`, Paperless will automatically tag your newly-consumed document with your `Home Utility` tag so long as the text `bc hydro` appears in the body of the document somewhere.

The matching logic is quite powerful, and supports searching the text of your document with different algorithms, and as such, some experimentation may be necessary to get things Just Right.

How Do I Set Up These Matching Algorithms?

Setting up of the algorithms is easily done through the admin interface. When you create a new correspondent or tag, there are optional fields for matching text and matching algorithm. From the help info there:

Note: Which algorithm you want to use when matching text to the OCR'd PDF. Here, "any" looks for any occurrence of any word provided in the PDF, while "all" requires that every word provided appear in the PDF, albeit not in the order provided. A "literal" match means that the text you enter must appear in the PDF exactly as you've entered it, and "regular expression" uses a regex to match the PDF. If you don't know what a regex is, you probably don't want this option.

When using the "any" or "all" matching algorithms, you can search for terms that consist of multiple words by enclosing them in double quotes. For example, defining a match text of `"Bank of America" BofA` using the "any" algorithm, will match documents that contain either "Bank of America" or "BofA", but will not match documents containing "Bank of South America".

Then just save your tag/correspondent and run another document through the consumer. Once complete, you should see the newly-created document, automatically tagged with the appropriate data.

2.7 Migrating, Updates, and Backups

As Paperless is still under active development, there's a lot that can change as software updates roll out. You should backup often, so if anything goes wrong during an update, you at least have a means of restoring to something usable. Thankfully, there are automated ways of backing up, restoring, and updating the software.

2.7.1 Backing Up

So you're bored of this whole project, or you want to make a remote backup of your files for whatever reason. This is easy to do, simply use the *exporter* to dump your documents and database out into an arbitrary directory.

2.7.2 Restoring

Restoring your data is just as easy, since nearly all of your data exists either in the file names, or in the contents of the files themselves. You just need to create an empty database (just follow the *installation instructions* again) and then import the `tags.json` file you created as part of your backup. Lastly, copy your exported documents into the consumption directory and start up the consumer.

```
$ cd /path/to/project
$ rm data/db.sqlite3  # Delete the database
$ cd src
$ ./manage.py migrate  # Create the database
$ ./manage.py createsuperuser
$ ./manage.py loaddata /path/to/arbitrary/place/tags.json
$ cp /path/to/exported/docs/* /path/to/consumption/dir/
$ ./manage.py document_consumer
```

Importing your data if you are *using Docker* is almost as simple:

```
# Stop and remove your current containers
$ docker-compose stop
$ docker-compose rm -f

# Recreate them, add the superuser
$ docker-compose up -d
$ docker-compose run --rm webserver createsuperuser

# Load the tags
$ cat /path/to/arbitrary/place/tags.json | docker-compose run --rm webserver loaddata_
↪stdin -

# Load your exported documents into the consumption directory
# (How you do this highly depends on how you have set this up)
$ cp /path/to/exported/docs/* /path/to/mounted/consumption/dir/
```

After loading the documents into the consumption directory the consumer will immediately start consuming the documents.

2.7.3 Updates

For the most part, all you have to do to update Paperless is run `git pull` on the directory containing the project files, and then use Django's `migrate` command to execute any database schema updates that might have been rolled in as part of the update:

```
$ cd /path/to/project
$ git pull
$ pip install -r requirements.txt
$ cd src
$ ./manage.py migrate
```

Note that it's possible (even likely) that while `git pull` may update some files, the `migrate` step may not update anything. This is totally normal.

Additionally, as new features are added, the ability to control those features is typically added by way of an environment variable set in `paperless.conf`. You may want to take a look at the `paperless.conf.example` file to see if there's anything new in there compared to what you've got int `/etc`.

If you are *using Docker* the update process is similar:

```
$ cd /path/to/project
$ git pull
$ docker build -t paperless .
$ docker-compose run --rm comsumer migrate
$ docker-compose up -d
```

If `git pull` doesn't report any changes, there is no need to continue with the remaining steps.

2.8 Customising Paperless

Currently, the Paperless' interface is just the default Django admin, which while powerful, is rather boring. If you'd like to give the site a bit of a face-lift, or if you simply want to adjust the colours, contrast, or font size to make things easier to read, you can do that by adding your own CSS or Javascript quite easily.

2.8.1 Overrides

On every page load, Paperless looks for two files in your media root directory (the directory defined by your `PAPERLESS_MEDIADIR` configuration variable or the default, `<project root>/media/`) for two files:

- `overrides.css`
- `overrides.js`

If it finds either or both of those files, they'll be loaded into the page: the CSS in the `<head>`, and the Javascript stuffed into the last line of the `<body>`.

An important note about customisation

Any changes you make to the site with your CSS or Javascript are likely to depend on the structure of the current HTML and/or the existing CSS rules. For the most part it's safe to assume that these bits won't change, but *sometimes they do* as features are added or bugs are fixed.

If you make a change that you think others would appreciate though, submit it as a pull request and maybe we can find a way to work it into the project by default!

2.9 Extending Paperless

For the most part, Paperless is monolithic, so extending it is often best managed by way of modifying the code directly and issuing a pull request on GitHub. However, over time the project has been evolving to be a little more "pluggable" so that users can write their own stuff that talks to it.

2.9.1 Parsers

You can leverage Paperless' consumption model to have it consume files *other* than ones handled by default like `.pdf`, `.jpg`, and `.tiff`. To do so, you simply follow Django's convention of creating a new app, with a few key requirements.

parsers.py

In this file, you create a class that extends `documents.parsers.DocumentParser` and go about implementing the three required methods:

- `get_thumbnail()`: Returns the path to a file we can use as a thumbnail for this document.
- `get_text()`: Returns the text from the document and only the text.
- `get_date()`: If possible, this returns the date of the document, otherwise it should return `None`.

signals.py

At consumption time, Paperless emits a `document_consumer_declaration` signal which your module has to react to in order to let the consumer know whether or not it's capable of handling a particular file. Think of it like this:

1. Consumer finds a file in the consumption directory.
2. It asks all the available parsers: *"Hey, can you handle this file?"*
3. Each parser responds with either `None` meaning they can't handle the file, or a dictionary in the following format:

```
{
    "parser": <the class name>,
    "weight": <an integer>
}
```

The consumer compares the `weight` values from all respondents and uses the class with the highest value to consume the document. The default parser, `RasterisedDocumentParser` has a weight of `0`.

apps.py

This is a standard Django file, but you'll need to add some code to it to connect your parser to the `document_consumer_declaration` signal.

Finally

The last step is to update `settings.py` to include your new module. Eventually, this will be dynamic, but at the moment, you have to edit the `INSTALLED_APPS` section manually. Simply add the path to your AppConfig to the list like this:

```
INSTALLED_APPS = [
    ...
    "my_module.apps.MyModuleConfig",
    ...
]
```

Order doesn't matter, but generally it's a good idea to place your module lower in the list so that you don't end up accidentally overriding project defaults somewhere.

An Example

The core Paperless functionality is based on this design, so if you want to see what a parser module should look like, have a look at parsers.py, signals.py, and apps.py in the paperless_tesseract module.

2.10 Troubleshooting

2.10.1 Consumer warns `OCR for XX failed`

If you find the OCR accuracy to be too low, and/or the document consumer warns that `OCR for XX failed, but we're going to stick with what we've got since FORGIVING_OCR is enabled`, then you might need to install the Tesseract language files marching your document's languages.

As an example, if you are running Paperless from any Ubuntu or Debian box, and your documents are written in Spanish you may need to run:

```
apt-get install -y tesseract-ocr-spa
```

2.10.2 Consumer dies with `convert: unable to extent pixel cache`

During the consumption process, Paperless invokes ImageMagick's `convert` program to translate the source document into something that the OCR engine can understand and this can burn a Very Large amount of memory if the original document is rather long. Similarly, if your system doesn't have a lot of memory to begin with (ie. a Raspberry Pi), then this can happen for even medium-sized documents.

The solution is to tell ImageMagick *not* to Use All The RAM, as is its default, and instead tell it to used a fixed amount. `convert` will then break up the job into hundreds of individual files and use them to slowly compile the finished image. Simply set `PAPERLESS_CONVERT_MEMORY_LIMIT` in `/etc/paperless.conf` to something like `32000000` and you'll limit `convert` to 32MB. Fiddle with this value as you like.

HOWEVER: Simply setting this value may not be enough on system where `/tmp` is mounted as tmpfs, as this is where `convert` will write its temporary files. In these cases (most Systemd machines), you need to tell ImageMagick to use a different space for its scratch work. You do this by setting `PAPERLESS_CONVERT_TMPDIR` in `/etc/paperless.conf` to somewhere that's actually on a physical disk (and writable by the user running Paperless), like `/var/tmp/paperless` or `/home/my_user/tmp` in a pinch.

2.10.3 DecompressionBombWarning and/or no text in the OCR output

Some users have had issues using Paperless to consume PDFs that were created by merging Very Large Scanned Images into one PDF. If this happens to you, it's likely because the PDF you've created contains some very large pages (millions of pixels) and the process of converting the PDF to a OCR-friendly image is exploding.

Typically, this happens because the scanned images are created with a high DPI and then rolled into the PDF with an assumed DPI of 72 (the default). The best solution then is to specify the DPI used in the scan in the conversion-to-PDF step. So for example, if you scanned the original image with a DPI of 300, then merging the images into the single PDF with `convert` should look like this:

```
$ convert -density 300 *.jpg finished.pdf
```

For more information on this and situations like it, you should take a look at Issue #118 as that's where this tip originated.

2.11 Contributing to Paperless

Maybe you've been using Paperless for a while and want to add a feature or two, or maybe you've come across a bug that you have some ideas how to solve. The beauty of Free software is that you can see what's wrong and help to get it fixed for everyone!

2.11.1 How to Get Your Changes Rolled Into Paperless

If you've found a bug, but don't know how to fix it, you can always post an issue on GitHub in the hopes that someone will have the time to fix it for you. If however you're the one with the time, pull requests are always welcome, you just have to make sure that your code conforms to a few standards:

Pep8

It's the standard for all Python development, so it's very well documented. The short version is:

- Lines should wrap at 79 characters
- Use `snake_case` for variables, `CamelCase` for classes, and `ALL_CAPS` for constants.
- Space out your operators: `stuff + 7` instead of `stuff+7`
- Two empty lines between classes, and functions, but 1 empty line between class methods.

There's more to it than that, but if you follow those, you'll probably be alright. When you submit your pull request, there's a pep8 checker that'll look at your code to see if anything is off. If it finds anything, it'll complain at you until you fix it.

Additional Style Guides

Where pep8 is ambiguous, I've tried to be a little more specific. These rules aren't hard-and-fast, but if you can conform to them, I'll appreciate it and spend less time trying to conform your PR before merging:

Function calls

If you're calling a function and that necessitates more than one line of code, please format it like this:

```
my_function(
    argument1,
    kwarg1="x",
    kwarg2="y"
    another_really_long_kwarg="some big value"
    a_kwarg_calling_another_long_function=another_function(
        another_arg,
        another_kwarg="kwarg!"
    )
)
```

This is all in the interest of code uniformity rather than anything else. If we stick to a style, everything is understandable in the same way.

Quoting Strings

pep8 is a little too open-minded on this for my liking. Python strings should be quoted with double quotes (") except in cases where the resulting string would require too much escaping of a double quote, in which case, a single quoted, or triple-quoted string will do:

```
my_string = "This is my string"
problematic_string = 'This is a "string" with "quotes" in it'
```

In HTML templates, please use double-quotes for tag attributes, and single quotes for arguments passed to Django tempalte tags:

```
<div class="stuff">
    <a href="{% url 'some-url-name' pk='w00t' %}">link this</a>
</div>
```

This is to keep linters happy they look at an HTML file and see an attribute closing the " before it should have been.

–

That's all there is in terms of guidelines, so I hope it's not too daunting.

Indentation & Spacing

When it comes to indentation:

- For Python, the rule is: follow pep8 and use 4 spaces.
- For Javascript, CSS, and HTML, please use 1 tab.

Additionally, Django templates making use of block elements like `{% if %}`, `{% for %}`, and `{% block %}` etc. should be indented:

Good:

```
{% block stuff %}
    <h1>This is the stuff</h1>
{% endblock %}
```

Bad:

```
{% block stuff %}
<h1>This is the stuff</h1>
{% endblock %}
```

2.11.2 The Code of Conduct

Paperless has a code of conduct. It's a lot like the other ones you see out there, with a few small changes, but basically it boils down to:

> Don't be an ass, or you might get banned.

I'm proud to say that the CoC has never had to be enforced because everyone has been awesome, friendly, and professional.

2.12 Scanner Recommendations

As Paperless operates by watching a folder for new files, doesn't care what scanner you use, but sometimes finding a scanner that will write to an FTP, NFS, or SMB server can be difficult. This page is here to help you find one that works right for you based on recommentations from other Paperless users.

Brand	Model	Supports			Recommended By
		FTP	NFS	SMB	
Brother	ADS-1500W	yes	no	yes	danielquinn
Brother	MFC-J6930DW	yes			ayounggun
Fujitsu	ix500	yes		yes	eonist

2.13 Changelog

2.13.1 2.6.0

- Allow an infinite number of logs to be deleted. Thanks to Ulli for noting the problem in #433.
- Fix the `RecentCorrespondentsFilter` correspondents filter that was added in 2.4 to play nice with the defaults. Thanks to tsia and Sblop who pointed this out. #423.
- Updated dependencies to include (among other things) a security patch to requests.

2.13.2 2.5.0

- **New dependency**: Paperless now optimises thumbnail generation with optipng, so you'll need to install that somewhere in your PATH or declare its location in `PAPERLESS_OPTIPNG_BINARY`. The Docker image has already been updated on the Docker Hub, so you just need to pull the latest one from there if you're a Docker user.
- "Login free" instances of Paperless were breaking whenever you tried to edit objects in the admin: adding/deleting tags or correspondents, or even fixing spelling. This was due to the "user hack" we were applying to sessions that weren't using a login, as that hack user didn't have a valid id. The fix was to attribute the first user id in the system to this hack user. #394
- A problem in how we handle slug values on Tags and Correspondents required a few changes to how we handle this field #393:
  1. Slugs are no longer editable. They're derived from the name of the tag or correspondent at save time, so if you wanna change the slug, you have to change the name, and even then you're restricted to the rules of the `slugify()` function. The slug value is still visible in the admin though.
  2. I've added a migration to go over all existing tags & correspondents and rewrite the `.slug` values to ones conforming to the `slugify()` rules.
  3. The consumption process now uses the same rules as `.save()` in determining a slug and using that to check for an existing tag/correspondent.
- An annoying bug in the date capture code was causing some bogus dates to be attached to documents, which in turn busted the UI. Thanks to Andrew Peng for reporting this. #414.
- A bug in the Dockerfile meant that Tesseract language files weren't being installed correctly. euri10 was quick to provide a fix: #406, #413.
- Document consumption is now wrapped in a transaction as per an old ticket #262.

- The `get_date()` functionality of the parsers has been consolidated onto the `DocumentParser` class since much of that code was redundant anyway.

2.13.3 2.4.0

- A new set of actions are now available thanks to jonaswinkler's very first pull request! You can now do nifty things like tag documents in bulk, or set correspondents in bulk. #405
- The import/export system is now a little smarter. By default, documents are tagged as `unencrypted`, since exports are by their nature unencrypted. It's now in the import step that we decide the storage type. This allows you to export from an encrypted system and import into an unencrypted one, or vice-versa.
- The migration history has been slightly modified to accommodate PostgreSQL users. Additionally, you can now tell paperless to use PostgreSQL simply by declaring `PAPERLESS_DBUSER` in your environment. This will attempt to connect to your Postgres database without a password unless you also set `PAPERLESS_DBPASS`.
- A bug was found in the REST API filter system that was the result of an update of django-filter some time ago. This has now been patched in #412. Thanks to thepill for spotting it!

2.13.4 2.3.0

- Support for consuming plain text & markdown documents was added by Joshua Taillon! This was a long-requested feature, and it's addition is likely to be greatly appreciated by the community: #395 Thanks also to David Martin for his assistance on the issue.
- dubit0 found & fixed a bug that prevented management commands from running before we had an operational database: #396
- Joshua also added a simple update to the thumbnail generation process to improve performance: #399
- As his last bit of effort on this release, Joshua also added some code to allow you to view the documents inline rather than download them as an attachment. #400
- Finally, ahyear found a slip in the Docker documentation and patched it. #401

2.13.5 2.2.1

- Kyle Lucy reported a bug quickly after the release of 2.2.0 where we broke the `DISABLE_LOGIN` feature: #392.

2.13.6 2.2.0

- Thanks to dadosch, Wolfgang Mader, and Tim Brooks this is the first version of Paperless that supports Django 2.0! As a result of their hard work, you can now also run Paperless on Python 3.7 as well: #386 & #390.
- Stéphane Brunner added a few lines of code that made tagging interface a lot easier on those of us with lots of different tags: #391.
- Kilian Koeltzsch noticed a bug in how we capture & automatically create tags, so that's fixed now too: #384.
- erikarvstedt tweaked the behaviour of the test suite to be better behaved for packaging environments: #383.
- Lukasz Soluch added CORS support to make building a new Javascript-based front-end cleaner & easier: #387.

2.13.7 2.1.0

- Enno Lohmeier added three simple features that make Paperless a lot more user (and developer) friendly:
  1. There's a new search box on the front page: #374.
  2. The correspondents & tags pages now have a column showing the number of relevant documents: #375.
  3. The Dockerfile has been tweaked to build faster for those of us who are doing active development on Paperless using the Docker environment: #376.
- You now also have the ability to customise the interface to your heart's content by creating a file called `overrides.css` and/or `overrides.js` in the root of your media directory. Thanks to Mark McFate for this idea: #371

2.13.8 2.0.0

This is a big release as we've changed a core-functionality of Paperless: we no longer encrypt files with GPG by default.

The reasons for this are many, but it boils down to that the encryption wasn't really all that useful, as files on-disk were still accessible so long as you had the key, and the key was most typically stored in the config file. In other words, your files are only as safe as the `paperless` user is. In addition to that, *the contents of the documents were never encrypted*, so important numbers etc. were always accessible simply by querying the database. Still, it was better than nothing, but the consensus from users appears to be that it was more an annoyance than anything else, so this feature is now turned off unless you explicitly set a passphrase in your config file.

Migrating from 1.x

Encryption isn't gone, it's just off for new users. So long as you have `PAPERLESS_PASSPHRASE` set in your config or your environment, Paperless should continue to operate as it always has. If however, you want to drop encryption too, you only need to do two things:

1. Run `./manage.py migrate && ./manage.py change_storage_type gpg unencrypted`. This will go through your entire database and Decrypt All The Things.
2. Remove `PAPERLESS_PASSPHRASE` from your `paperless.conf` file, or simply stop declaring it in your environment.

Special thanks to erikarvstedt, matthewmoto, and mcronce who did the bulk of the work on this big change.

2.13.9 1.4.0

- Quentin Dawans has refactored the document consumer to allow for some command-line options. Notably, you can now direct it to consume from a particular `--directory`, limit the `--loop-time`, set the time between mail server checks with `--mail-delta` or just run it as a one-off with `--one-shot`. See #305 & #313 for more information.
- Refactor the use of travis/tox/pytest/coverage into two files: `.travis.yml` and `setup.cfg`.
- Start generating requirements.txt from a Pipfile. I'll probably switch over to just using pipenv in the future.
- All for a alternative FreeBSD-friendly location for `paperless.conf`. Thanks to Martin Arendtsen who provided this (#322).
- Document consumption events are now logged in the Django admin events log. Thanks to CkuT for doing the legwork on this one and to Quentin Dawans & David Martin for helping to coordinate & work out how the feature would be developed.

- erikarvstedt contributed a pull request (#328) to add `--noreload` to the default server start process. This helps reduce the load imposed by the running webservice.
- Through some discussion on #253 and #323, we've removed a few of the hardcoded URL values to make it easier for people to host Paperless on a subdirectory. Thanks to Quentin Dawans and Kyle Lucy for helping to work this out.
- The clickable area for documents on the listing page has been increased to a more predictable space thanks to a glorious hack from erikarvstedt in #344.
- Strubbl noticed an annoying bug in the bash script wrapping the Docker entrypoint and fixed it with some very creating Bash skills: #352.
- You can now use the search field to find documents by tag thanks to thinkjk's *first ever issue*: #354.
- Inotify is now being used to detect additions to the consume directory thanks to some excellent work from erikarvstedt on #351

2.13.10 1.3.0

- You can now run Paperless without a login, though you'll still have to create at least one user. This is thanks to a pull-request from matthewmoto: #295. Note that logins are still required by default, and that you need to disable them by setting `PAPERLESS_DISABLE_LOGIN="true"` in your environment or in `/etc/paperless.conf`.
- Fix for #303 where sketchily-formatted documents could cause the consumer to break and insert half-records into the database breaking all sorts of things. We now capture the return codes of both `convert` and `unpaper` and fail-out nicely.
- Fix for additional date types thanks to input from Isaac and code from BastianPoe (#301).
- Fix for running migrations in the Docker container (#299). Thanks to Georgi Todorov for the fix (#300) and to Pit for the review.
- Fix for Docker cases where the issuing user is not UID 1000. This was a collaborative fix between Jeffrey Portman and Pit in #311 and #312 to fix #306.
- Patch the historical migrations to support MySQL's um, *interesting* way of handing indexes (#308). Thanks to Simon Taddiken for reporting the problem and helping me find where to fix it.

2.13.11 1.2.0

- New Docker image, now based on Alpine, thanks to the efforts of addadi and Pit. This new image is dramatically smaller than the Debian-based one, and it also has a new home on Docker Hub. A proper thank-you to Pit for hosting the image on his Docker account all this time, but after some discussion, we decided the image needed a more *official-looking* home.
- BastianPoe has added the long-awaited feature to automatically skip the OCR step when the PDF already contains text. This can be overridden by setting `PAPERLESS_OCR_ALWAYS=YES` either in your `paperless.conf` or in the environment. Note that this also means that Paperless now requires `libpoppler-cpp-dev` to be installed. **Important**: You'll need to run `pip install -r requirements.txt` after the usual `git pull` to properly update.
- BastianPoe has also contributed a monumental amount of work (#291) to solving #158: setting the document creation date based on finding a date in the document text.

2.13.12 1.1.0

- Fix for #283, a redirect bug which broke interactions with paperless-desktop. Thanks to chris-aeviator for reporting it.
- Addition of an optional new financial year filter, courtesy of David Martin #256
- Fixed a typo in how thumbnails were named in exports #285, courtesy of Dan Panzarella

2.13.13 1.0.0

- Upgrade to Django 1.11. **You'll need to run ''pip install -r requirements.txt'' after the usual ''git pull'' to properly update**.
- Replace the templatetag-based hack we had for document listing in favour of a slightly less ugly solution in the form of another template tag with less copypasta.
- Support for multi-word-matches for auto-tagging thanks to an excellent patch from ishirav #277.
- Fixed a CSS bug reported by Stefan Hagen that caused an overlapping of the text and checkboxes under some resolutions #272.
- Patched the Docker config to force the serving of static files. Credit for this one goes to dev-rke via #248.
- Fix file permissions during Docker start up thanks to Pit on #268.
- Date fields in the admin are now expressed as HTML5 date fields thanks to Lukas Winkler's issue #278

2.13.14 0.8.0

- Paperless can now run in a subdirectory on a host (`/paperless`), rather than always running in the root (`/`) thanks to maphy-psd's work on #255.

2.13.15 0.7.0

- **Potentially breaking change**: As per #235, Paperless will no longer automatically delete documents attached to correspondents when those correspondents are themselves deleted. This was Django's default behaviour, but didn't make much sense in Paperless' case. Thanks to Thomas Brueggemann and David Martin for their input on this one.
- Fix for #232 wherein Paperless wasn't recognising `.tif` files properly. Thanks to ayounggun for reporting this one and to Kusti Skytén for posting the correct solution in the Github issue.

2.13.16 0.6.0

- Abandon the shared-secret trick we were using for the POST API in favour of BasicAuth or Django session.
- Fix the POST API so it actually works. #236
- **Breaking change**: We've dropped the use of `PAPERLESS_SHARED_SECRET` as it was being used both for the API (now replaced with a normal auth) and form email polling. Now that we're only using it for email, this variable has been renamed to `PAPERLESS_EMAIL_SECRET`. The old value will still work for a while, but you should change your config if you've been using the email polling feature. Thanks to Joshua Gilman for all the help with this feature.

2.13.17 0.5.0

- Support for fuzzy matching in the auto-tagger & auto-correspondent systems thanks to Jake Gysland's patch #220.
- Modified the Dockerfile to prepare an export directory (#212). Thanks to combined efforts from Pit and Strubbl in working out the kinks on this one.
- Updated the import/export scripts to include support for thumbnails. Big thanks to CkuT for finding this shortcoming and doing the work to get it fixed in #224.
- All of the following changes are thanks to David Martin: * Bumped the dependency on pyocr to 0.4.7 so new users can make use of Tesseract 4 if they so prefer (#226). * Fixed a number of issues with the automated mail handler (#227, #228) * Amended the documentation for better handling of systemd service files (#229) * Amended the Django Admin configuration to have nice headers (#230)

2.13.18 0.4.1

- Fix for #206 wherein the pluggable parser didn't recognise files with all-caps suffixes like `.PDF`

2.13.19 0.4.0

- Introducing reminders. See #199 for more information, but the short explanation is that you can now attach simple notes & times to documents which are made available via the API. Currently, the default API (basically just the Django admin) doesn't really make use of this, but Thomas Brueggemann over at Paperless Desktop has said that he would like to make use of this feature in his project.

2.13.20 0.3.6

- Fix for #200 (!!) where the API wasn't configured to allow updating the correspondent or the tags for a document.
- The `content` field is now optional, to allow for the edge case of a purely graphical document.
- You can no longer add documents via the admin. This never worked in the first place, so all I've done here is remove the link to the broken form.
- The consumer code has been heavily refactored to support a pluggable interface. Install a paperless consumer via pip and tell paperless about it with an environment variable, and you're good to go. Proper documentation is on its way.

2.13.21 0.3.5

- A serious facelift for the documents listing page wherein we drop the tabular layout in favour of a tiled interface.
- Users can now configure the number of items per page.
- Fix for #171: Allow users to specify their own `SECRET_KEY` value.
- Moved the dotenv loading to the top of settings.py
- Fix for #112: Added checks for binaries required for document consumption.

2.13.22 0.3.4

- Removal of django-suit due to a licensing conflict I bumped into in 0.3.3. Note that you *can* use Django Suit with Paperless, but only in a non-profit situation as their free license prohibits for-profit use. As a result, I can't bundle Suit with Paperless without conflicting with the GPL. Further development will be done against the stock Django admin.
- I shrunk the thumbnails a little 'cause they were too big for me, even on my high-DPI monitor.
- BasicAuth support for document and thumbnail downloads, as well as the Push API thanks to @thomasbruegge-mann. See #179.

2.13.23 0.3.3

- Thumbnails in the UI and a Django-suit -based face-lift courtesy of @ekw!
- Timezone, items per page, and default language are now all configurable, also thanks to @ekw.

2.13.24 0.3.2

- Fix for #172: defaulting ALLOWED_HOSTS to `["*"]` and allowing the user to set her own value via `PAPERLESS_ALLOWED_HOSTS` should the need arise.

2.13.25 0.3.1

- Added a default value for `CONVERT_BINARY`

2.13.26 0.3.0

- Updated to using django-filter 1.x
- Added some system checks so new users aren't confused by misconfigurations.
- Consumer loop time is now configurable for systems with slow writes. Just set `PAPERLESS_CONSUMER_LOOP_TIME` to a number of seconds. The default is 10.
- As per #44, we've removed support for `PAPERLESS_CONVERT`, `PAPERLESS_CONSUME`, and `PAPERLESS_SECRET`. Please use `PAPERLESS_CONVERT_BINARY`, `PAPERLESS_CONSUMPTION_DIR`, and `PAPERLESS_SHARED_SECRET` respectively instead.

2.13.27 0.2.0

- #150: The media root is now a variable you can set in `paperless.conf`.
- #148: The database location (sqlite) is now a variable you can set in `paperless.conf`.
- #146: Fixed a bug that allowed unauthorised access to the `/fetch` URL.
- #131: Document files are now automatically removed from disk when they're deleted in Paperless.
- #121: Fixed a bug where Paperless wasn't setting document creation time based on the file naming scheme.
- #81: Added a hook to run an arbitrary script after every document is consumed.

- #98: Added optional environment variables for ImageMagick so that it doesn't explode when handling Very Large Documents or when it's just running on a low-memory system. Thanks to Florian Harr for his help on this one.
- #89 Ported the auto-tagging code to correspondents as well. Thanks to Justin Snyman for the pointers in the issue queue.
- Added support for guessing the date from the file name along with the correspondent, title, and tags. Thanks to Tikitu de Jager for his pull request that I took forever to merge and to Pit for his efforts on the regex front.
- #94: Restored support for changing the created date in the UI. Thanks to Martin Honermeyer and Tim White for working with me on this.

2.13.28 0.1.1

- Potentially **Breaking Change**: All references to "sender" in the code have been renamed to "correspondent" to better reflect the nature of the property (one could quite reasonably scan a document before sending it to someone.)
- #67: Rewrote the document exporter and added a new importer that allows for full metadata retention without depending on the file name and modification time. A big thanks to Tikitu de Jager, Pit, Florian Jung, and Christopher Luu for their code snippets and contributing conversation that lead to this change.
- #20: Added *unpaper* support to help in cleaning up the scanned image before it's OCR'd. Thanks to Pit for this one.
- #71 Added (encrypted) thumbnails in anticipation of a proper UI.
- #68: Added support for using a proper config file at `/etc/paperless.conf` and modified the systemd unit files to use it.
- Refactored the Vagrant installation process to use environment variables rather than asking the user to modify `settings.py`.
- #44: Harmonise environment variable names with constant names.
- #60: Setup logging to actually use the Python native logging framework.
- #53: Fixed an annoying bug that caused `.jpeg` and `.JPG` images to be imported but made unavailable.

2.13.29 0.1.0

- Docker support! Big thanks to Wayne Werner, Brian Conn, and Tikitu de Jager for this one, and especially to Pit who spearheadded this effort.
- A simple REST API is in place, but it should be considered unstable.
- Cleaned up the consumer to use temporary directories instead of a single scratch space. (Thanks Pit)
- Improved the efficiency of the consumer by parsing pages more intelligently and introducing a threaded OCR process (thanks again Pit).
- #45: Cleaned up the logic for tag matching. Reported by darkmatter.
- #47: Auto-rotate landscape documents. Reported by Paul and fixed by Pit.
- #48: Matching algorithms should do so on a word boundary (darkmatter)
- #54: Documented the re-tagger (zedster)
- #57: Make sure file is preserved on import failure (darkmatter)

- Added tox with pep8 checking

2.13.30 0.0.6

- Added support for parallel OCR (significant work from Pit)
- Sped up the language detection (significant work from Pit)
- Added simple logging

2.13.31 0.0.5

- Added support for image files as documents (png, jpg, gif, tiff)
- Added a crude means of HTTP POST for document imports
- Added IMAP mail support
- Added a re-tagging utility
- Documentation for the above as well as data migration

2.13.32 0.0.4

- Added automated tagging basted on keyword matching
- Cleaned up the document listing page
- Removed `User` and `Group` from the admin
- Added `pytz` to the list of requirements

2.13.33 0.0.3

- Added basic tagging

2.13.34 0.0.2

- Added language detection
- Added datestamps to `document_exporter`.
- Changed `settings.TESSERACT_LANGUAGE` to `settings.OCR_LANGUAGE`.

2.13.35 0.0.1

- Initial release