PRACTICAL PORTS

Go Paperless

BY BENEDICT REUSCHLING

During the height of the Pandemic, I was staying home and the days were basically like any other. Time passed slowly, and looking around my room, I became painfully aware of how untidy it had become. Especially around my work desk, where piles of books, notes, letters and other pieces of paper had started a race to the top. I decided to do something about it. I put on a podcast and started picking up, ordering, throwing away, and cleaning my way toward a clean desk top. This was a great feeling of accomplishment in times of a global crisis. Motivated by that, I decided to start scanning those papers to reduce the pile.

I had purchased a mobile scanner a while ago, which connects via USB and uses some proprietary, yet efficient, scanning software. The scanner was not smaller than a rolling pin, and a single piece of paper would fit. Since I had time, I continued scanning each paper, giving the resulting PDF a description and date, moving on to the next one. This was tedious, but I got through it in the end. Ordering them afterwards was another long task: taxes, insurance, contracts, receipts, bills (both sent and received), records, certificates, and more needed to end up in the proper directories.

This software re-scans PDFs and makes the text into something that one can extract and search individual words.

The scanning software was only 32-bit, and the entire scanning unit stopped working a couple of years later — just when another pile of papers was dangerously close to tumbling over. It was time to look for alternatives. I had found `textproc/py-ocrmypdf` a few months earlier. Essentially, this software re-scans PDFs and makes the text, which is sometimes a big picture, into something that one can extract and search individual words. It uses pattern recognition to figure out the words and language and the results are surprisingly good. I ran this over my existing scans and now I can do a full-text search over a document and find how expensive my flight to BSDCan 2014 had been, for example.

Introducing Paperless-ngx

Then I found paperless-ngx, which `ocrmypdf` is a part of, but a comparably small part of a greater whole. It is a document management system that you can pass documents into and it creates an archive that is fully searchable, automatically detects the content (AI is everywhere) and files it away according to rules you define. Basically, I can feed it a letter, and it figures out that it comes from my bank, files it away based on the date it detected, while **ocrmypdf**-ing it makes it searchable and adds useful metadata as well. The file ends up in a directory that can either be subject related (everything I ever got from that particular bank), or by year-month-date-description, or something completely up to you. You can also feed entire directories into paperless-ngx, and it figures out which documents it has already scanned and skips those, while the rest run through a processing pipeline. With each document, the chances grow that future documents like it get properly categorized. Plus, it comes with a nice web UI into which you can drag and drop files for scanning and find existing documents with ease. Another way to feed documents into it is via an "incoming" folder that you can share among colleagues in an office or by sending documents as attachments (remember emails?) to it.

With each document, the chances grow that future documents like it get properly categorized.

The software stack itself is impressive and may be too intimidating, even with the excellent documentation that the paperless-ngx website provides. Plenty of software and services have to work together to get a smooth scanning experience. Luckily, there is a port for it under **deskutils/py-paperless-ngx.** Even better, the port maintainer created a post-install man page detailing all the steps to get a working paperless-ngx stack going. Did I mention that I love ports maintainers? With these instructions, I was able to set up my own paperless-ngx in no time. First on a Raspberry Pi 3 and then on a Pi 4. It works, even though the Pi 3 stretches your patience because of the required processing power to get the final result. With the Pi 4 though, I've had good experiences and the scanning time is decent enough. You could run this in the office or at home with a negligible footprint on your electricity bill, while allowing other people to scan their docs without seeing those of others. If you're dealing with a lot of documents and want to have them digitized, take a look at the paperless-ngx setup we're doing here. You can thank me later...

Paperless-ngx Setup

Whether you are using a Raspberry Pi, a different embedded device, or a full-blown server does not really matter. As long as it runs FreeBSD, you can follow along. I'm not spending any time on the base installation or hardening the system, as there are plenty of other good articles available that cover that. Just make sure to do exactly that when you connect your paperless-ngx service to the network for other people to use.

Start by installing the paperless-ngx port:

```
# pkg install deskutils/py-paperless-ngx
```

You'll be greeted by the **pkg-message** after installation, advising you to take a look at the man page for further instructions. Without them, you have only the basic service, which does not do too much at this point.

Most files end up in **/var/db/paperless**, which you can probably put on a separate ZFS dataset, but in my experience, the compression savings are not worth it. But your mileage may vary and ZFS is generally a good idea for storing those precious documents.

Paperless-ngx wants to have access to a Redis instance, which is what we're installing next:

```
# pkg install redis
# service redis enable
# service redis start
```

Easy enough, having it both installed and started at boot time, as well as the current session with these three commands. If you have Redis running somewhere else in your network, you need to modify and add its credentials to `/usr/local/etc/paperless.conf`. When running on localhost, it's fine to run it without any special privileges since it won't be reachable from other hosts this way.

The configuration file is well documented with comments. Some items like `THREADS_PER_WORKER` (mine is at 1 on the RPI 4), `PAPERLESS_URL` (IP address or DNS name), and `PAPERLESS_TIME_ZONE` (I use UTF) should be modified to fit your system and network. Many other settings are fine in their defaults for your first couple of scans. You can always revisit this file and make modifications later.

Paperless-ngx is backed by a database to store various information. It's as easy to initialize as you can imagine using the following command:

```
# service paperless-migrate onestart
```

If you want to run this every time the system starts, you can execute this `service` command as well:

```
# service paperless-migrate enable
```

After that is done, we will start the backend services that paperless uses in order:

```
# service paperless-beat enable
# service paperless-consumer enable
# service paperless-webui enable
# service paperless-worker enable
```

You can find individual descriptions of these on the paperless-ngx website. Since we want to use paperless-ngx without restarting the system, we start all these services next:

```
# service paperless-beat start
# service paperless-consumer start
# service paperless-webui start
# service paperless-worker start
```

Machine learning is all the rage behind the AI hype. Paperless-ngx uses it as well, but mostly to aid in the character recognition to figure out the language of the document at hand. To do that, it uses the Natural Language Toolkit (NLTK). To download the necessary files, the following one-liner does the trick (replace the python version if necessary):

```
# su -l paperless -c '/usr/local/bin/python3.11 -m nltk.downloader \
 stopwords snowball_data punkt -d /var/db/paperless/nltkdata'
```

Documents are classified in different ways, which is the responsibility of the Celery component. This classification is done automatically upon scanning, but you can trigger it manually with this invocation:

```
# su -l paperless -c '/usr/local/bin/paperless document_create_classifier'
```

Celery also runs an optional component called Flower. It monitors a cluster of workers that Celery controls. This is an optional component and I run my instance without it. But for those who want all the bells and whistles, here is how to start it:

```
# service paperless-flower enable
# service paperless-flower start
```

Setting up the Web UI

To protect your Django-based Web UI holding all your documents scanned so far, you can set a superuser password like this:

```
# su -l paperless -c '/usr/local/bin/paperless createsuperuser'
```

I run an nginx webserver already (SSL proxy), so I can re-use that to point to my paperless-ngx website. If you don't have one already, the port also provides a ready-to-use configuration file in **`/usr/local/share/examples/paperless-ngx/nginx.conf`** that you just have to copy into your **`/usr/local/etc/nginx/`** directory. This includes an SSL configuration as well to not let people sniffing traffic figure out the login and doing other nasty things. To create a key that's valid for a whole year, run this lengthy **`openssl`** incantation (or get a key via **`lets-encrypt`**):

```
# openssl req -x509 -nodes -days 365 -newkey rsa:4096 \
 -keyout /usr/local/etc/nginx/selfsigned.key \
 -out /usr/local/etc/nginx/selfsigned.crt
```

Of course, you can make your own adjustments to the **`nginx.conf`** when necessary. When finished, enable it to start at boot time and for the current session:

```
# service nginx enable
# service nginx start
```

Voila! Now point your browser to the web URL defined in the **`paperless.conf`** and log into the application.

Basic configuration in the web UI

Before scanning your first document, I would recommend setting up a couple of items in the "Manage" section on the left, first. To begin, Correspondents are people or organizations that have sent you the paper. Think of banks, insurance companies, but also individuals. You can give them a descriptive name and configure paperless-ngx as to whether it should file a document with this correspondent if it detects certain keywords or other criteria.

Next, define document types. A contract is different from a love letter, which differs from a bill, which is not the same as certificate, and so on. This way, you can let paperless-ngx distinguish whether someone has sent you a bill or if that same person gave you a contract. Both can happen, and especially government agencies have a tendency (at least where I live) to correspond with you in different contexts, which you want to keep separate from each other. That's where paperless-ngx shines: once you defined your most active Correspondents and their typical documents, you don't need to worry about the proper classification anymore. Simply add documents and let paperless-ngx do it's work. With a bit of tweaking, you can scan a whole bunch of documents. But how do you order them? That's where storage paths come in.

These paths define where in your filesystem the documents should end up and under which directory hierarchy. I personally use `{created_year}/{correspondent}/{title}`, which means I have directories like 2024/insuranceXZY/YearlyReport.pdf. If you want to file all tax-related documents in a separate directory, define that under the storage paths section and define a rule to match when a document fits that criteria. The best part is if you change your mind about the ordering, changing the storage paths will automatically move and rename your already scanned documents within the file system without you doing a lengthy `mkdir`, `cp`, `mv`, `rm` dance.

But how do you order them? That's where storage paths come in.

Ready, set, scan

That's all for now. Drag a PDF document that you have lying around onto the web UI and see paperless-ngx start processing it. The Logs section on the left has details on how paperless-ngx choses to match correspondents and other details, which help you fine-tune your match rules. After processing is done, you can find the final result on the Dashboard or in the Documents folder. Continue scanning some more documents. They'll all end up in the `/var/db/paperless/media/documents/archive` directory (if you have not changed it in the `paperless.conf`), followed by the storage paths definition. I hope you'll find paperless-ngx as useful for your documents as I do. I'm always looking forward to the next letter I receive just to scan it with paperless-ngx. Thanks to the people creating paperless-ngx and those who made the FreeBSD port such a great installation experience.

BENEDICT REUSCHLING is a documentation committer in the FreeBSD project and member of the documentation engineering team. In the past, he served on the FreeBSD core team for two terms. He administers a big data cluster at the University of Applied Sciences, Darmstadt, Germany. He's also teaching a course "Unix for Developers" for undergraduates. Benedict is one of the hosts of the weekly bsdnow.tv podcast.