ocrmypdf Documentation

Release 8.0.1

James R. Barlow

2019-01-25

Contents

OCRmyPDF adds an OCR text layer to scanned PDF files, allowing them to be searched.

PDF is the best format for storing and exchanging scanned documents. Unfortunately, PDFs can be difficult to modify. OCRmyPDF makes it easy to apply image processing and OCR to existing PDFs.

CHAPTER 1

Introduction

OCRmyPDF is a Python 3 package that adds OCR layers to PDFs.

1.1 About OCR

Optical character recognition is technology that converts images of typed or handwritten text, such as in a scanned document, to computer text that can be searched and copied.

OCRmyPDF uses Tesseract, the best available open source OCR engine, to perform OCR.

1.2 About PDFs

PDFs are page description files that attempts to preserve a layout exactly. They contain vector graphics that can contain raster objects such as scanned images. Because PDFs can contain multiple pages (unlike many image formats) and can contain fonts and text, it is a good formats for exchanging scanned documents.

A PDF page might contain multiple images, even if it only appears to have one image. Some scanners or scanning software will segment pages into monochromatic text and color regions for example, to improve the compression ratio and appearance of the page.

Rasterizing a PDF is the process of generating an image suitable for display or analyzing with an OCR engine. OCR engines like Tesseract work with images, not vector objects.

1.3 About PDF/A

PDF/A is an ISO-standardized subset of the full PDF specification that is designed for archiving (the 'A' stands for Archive). PDF/A differs from PDF primarily by omitting features that would make it difficult to read the file in the future, such as embedded Javascript, video, audio and references to external fonts. All fonts and resources needed

to interpret the PDF must be contained within it. Because PDF/A disables Javascript and other types of embedded content, it is probably more secure.

There are various conformance levels and versions, such as "PDF/A-2b".

Generally speaking, the best format for scanned documents is PDF/A. Some governments and jurisdictions, US Courts in particular, mandate the use of PDF/A for scanned documents.

Since most people who scan documents are interested in reading them indefinitely into the future, OCRmyPDF generates PDF/A-2b by default.

PDF/A has a few drawbacks. Some PDF viewers include an alert that the file is a PDF/A, which may confuse some users. It also tends to produce larger files than PDF, because it embeds certain resources even if they are commonly available. PDF/A files can be digitally signed, but may not be encrypted, to ensure they can be read in the future. Fortunately, converting from PDF/A to a regular PDF is trivial, and any PDF viewer can view PDF/A.

1.4 What OCRmyPDF does

OCRmyPDF analyzes each page of a PDF to determine the colorspace and resolution (DPI) needed to capture all of the information on that page without losing content. It uses Ghostscript to rasterize the page, and then performs on OCR on the rasterized image to create an OCR "layer". The layer is then grafted back onto the original PDF.

While one can use a program like Ghostscript or ImageMagick to get an image and put the image through Tesseract, that actually creates a new PDF and many details may be lost. OCRmyPDF can produce a minimally changed PDF as output.

OCRmyPDF also some image processing options like deskew which improve the appearance of files and quality of OCR. When these are used, the OCR layer is grafted onto the processed image instead.

By default, OCRmyPDF produces archival PDFs – PDF/A, which are a stricter subset of PDF features designed for long term archives. If regular PDFs are desired, this can be disabled with `--output-type pdf`.

1.5 Why you shouldn't do this manually

A PDF is similar to an HTML file, in that it contains document structure along with images. Sometimes a PDF does nothing more than present a full page image, but often there is additional content that would be lost.

A manual process could work like either of these:

1. Rasterize each page as an image, OCR the images, and combine the output into a PDF. This preserves the layout of each page, but resamples all images (possibly losing quality, increasing file size, introducing compression artifacts, etc.).
2. Extract each image, OCR, and combine the output into a PDF. This loses the context in which images are used in the PDF, meaning that cropping, rotation and scaling of pages may be lost. Some scanned PDFs use multiple images segmented into black and white, grayscale and color regions, with stencil masks to prevent overlap, as this can enhance the appearance of a file while reducing file size. Clearly, reassembling these images will be easy. This also loses and text or vector art on any pages in a PDF with both scanned and pure digital content.

In the case of a PDF that is nothing other than a container of images (no rotation, scaling, cropping, one image per page), the second approach can be lossless.

OCRmyPDF uses several strategies depending on input options and the input PDF itself, but generally speaking it rasterizes a page for OCR and then grafts the OCR back onto the original. As such it can handle complex PDFs and still preserve their contents as much as possible.

OCRmyPDF also supports a many, many edge cases that have cropped over several years of development. We support PDF features like images inside of Form XObjects, and pages with UserUnit scaling. We support rare image formats like non-monochrome 1-bit images. We warn about files you may not to OCR. Thanks to pikepdf and QPDF, we auto-repair PDFs that are damaged. (Not that you need to know what any of these are! You should be able to throw any PDF at it.)

1.6 Limitations

OCRmyPDF is limited by the Tesseract OCR engine. As such it experiences these limitations, as do any other programs that rely on Tesseract:

- The OCR is not as accurate as commercial solutions such as Abbyy.
- It is not capable of recognizing handwriting.
- It may find gibberish and report this as OCR output.
- If a document contains languages outside of those given in the `-l LANG` arguments, results may be poor.
- It is not always good at analyzing the natural reading order of documents. For example, it may fail to recognize that a document contains two columns, and may try to join text across columns.
- Poor quality scans may produce poor quality OCR. Garbage in, garbage out.
- It does not expose information about what font family text belongs to.

OCRmyPDF is also limited by the PDF specification:

- PDF encodes the position of text glyphs but does not encode document structure. There is no markup that divides a document in sections, paragraphs, sentences, or even words (since blank spaces are not represented). As such all elements of document structure including the spaces between words must be derived heuristically. Some PDF viewers do a better job of this than others.
- Because some popular open source PDF viewers have a particularly hard time with spaces betweem words, OCRmyPDF appends a space to each text element as a workaround (when using `--pdf-renderer hocr`). While this mixes document structure with graphical information that ideally should be left to the PDF viewer to interpret, it improves compatibility with some viewers and does not cause problems for better ones.

Ghostscript also imposes some limitations:

- PDFs containing JBIG2-encoded content will be converted to CCITT Group4 encoding, which has lower compression ratios, if Ghostscript PDF/A is enabled.
- PDFs containing JPEG 2000-encoded content will be converted to JPEG encoding, which may introduce compression artifacts, if Ghostscript PDF/A is enabled.
- Ghostscript may transcode grayscale and color images, either lossy to lossless or lossless to lossy, based on an internal algorithm. This behavior can be suppressed by setting `--pdfa-image-compression` to `jpeg` or `lossless` to set all images to one type or the other. Ghostscript has no option to maintain the input image's format. (Ghostscript 9.25+ can copy JPEG images without transcoding them; earlier versions will transcode.)
- Ghostscript's PDF/A conversion removes any XMP metadata that is not one of the standard XMP metadata namespaces for PDFs. In particular, PRISM Metdata is removed.

Regarding OCRmyPDF itself:

- PDFs that use transparency are not currently represented in the test suite
- The Python API exported by `import ocrmypdf` is design to help scripts that use OCRmyPDF but is not currently capable of running OCRmyPDF jobs due to limitations in an underlying library.

1.7 Similar programs

To the author's knowledge, OCRmyPDF is the most feature-rich and thoroughly tested command line OCR PDF conversion tool. If it does not meet your needs, contributions and suggestions are welcome. If not, consider one of these similar open source programs:

- pdf2pdfocr
- pdfsandwich
- pypdfocr
- pdfbeads

1.8 Web front-ends

The Docker image ocrmypdf-webservice provides a web service front-end that allows files to submitted over HTTP and the results "downloaded". This is an HTTP server intended to simplify web services deployments; it is not intended to be deployed on the public internet and no real security measures to speak of.

In addition, the following integrations are available:

- Nextcloud OCR is a free software plugin for the Nextcloud private cloud software

Bear in mind that OCRmyPDF is not designed to be secure against malware-bearing PDFs (see **'Using OCRmyPDF online'**_). Users should ensure they comply with OCRmyPDF's licenses and the licenses of all dependencies. In particular, OCRmyPDF requires Ghostscript, which is licensed under AGPLv3.

CHAPTER 2

Release notes

OCRmyPDF uses semantic versioning for its command line interface and its public API.

The `ocrmypdf` package may now be imported. The public API may be useful in scripts that launch OCRmyPDF processes or that wish to use some of its features for working with PDFs.

Unfortunately, the public API does **not** expose the ability to actually OCR a PDF. This is due to a limitation in an underlying library (ruffus) that makes OCRmyPDF non-reentrant.

Note that it is licensed under GPLv3, so scripts that `import ocrmypdf` and are released publicly should probably also be licensed under GPLv3.

2.1 v8.0.1

- Fixed an exception when parsing PDFs that are missing a required field. #325
- pikepdf 1.0.5 is now required, to address some other PDF parsing issues.

2.2 v8.0.0

No major features. The intent of this release is to sever support for older versions of certain dependencies.

Breaking changes

- Dropped support for Tesseract 3.x. Tesseract 4.0 or newer is now required.
- Dropped support for Python 3.5.
- Some `ocrmypdf.pdfa` APIs that were deprecated in v7.x were removed. This functionality has been moved to pikepdf.

Other changes

- Fixed an unhandled exception when attempting to mask barcodes. #322

- It is now possible to use ocrmypdf without pdfminer.six, to support distributions that do not have it or cannot currently use it (e.g. Homebrew). Downstream maintainers should include pdfminer.six if possible.
- A warning is now issue when PDF/A conversion removes some XMP metadata from the input PDF. (Only a "whitelist" of certain XMP metadata types are allowed in PDF/A.)
- Fixed several issues that caused PDF/As to be produced with nonconforming XMP metadata (would fail validation with veraPDF).
- Fixed some instances where invalid DocumentInfo from a PDF cause XMP metadata creation to fail.
- Fixed a few documentation problems.
- pikepdf 1.0.2 is now required.

2.3 v7.4.0

- `--force-ocr` may now be used with the new `--threshold` and `--mask-barcodes` features
- pikepdf >= 0.9.1 is now required.
- Changed metadata handling to pikepdf 0.9.1. As a result, metadata handling of non-ASCII characters in Ghostscript 9.25 or later is fixed.
- chardet >= 3.0.4 is temporarily listed as required. pdfminer.six depends on it, but the most recent release does not specify this requirement. (#326)
- python-xmp-toolkit and libexempi are no longer required.
- A new Docker image is now being provided for users who wish to access OCRmyPDF over a simple HTTP interface, instead of the command line.
- Increase tolerance of PDFs that overflow or underflow the PDF graphics stack. (#325)

2.4 v7.3.1

- Fixed performance regression from v7.3.0; fast page analysis was not selected when it should be.
- Fixed a few exceptions related to the new `--mask-barcodes` feature and improved argument checking
- Added missing detection of TrueType fonts that lack a Unicode mapping

2.5 v7.3.0

- Added a new feature `--redo-ocr` to detect existing OCR in a file, remove it, and redo the OCR. This may be particularly helpful for anyone who wants to take advantage of OCR quality improvements in Tesseract 4.0. Note that OCR added by OCRmyPDF before version 3.0 cannot be detected since it was not properly marked as invisible text in the earliest versions. OCR that constructs a font from visible text, such as Adobe Acrobat's ClearScan.
- OCRmyPDF's content detection is generally more sophisticated. It learns more about the contents of each PDF and makes better recommendations:
  - OCRmyPDF can now detect when a PDF contains text that cannot be mapped to Unicode (meaning it is readable to human eyes but copy-pastes as gibberish). In these cases it recommends `--force-ocr` to make the text searchable.

  - PDFs containing vector objects are now rendered at more appropriate resolution for OCR.
  - We now exit with an error for PDFs that contain Adobe LiveCycle Designer's dynamic XFA forms. Currently the open source community does not have tools to work with these files.
  - OCRmyPDF now warns when a PDF that contains Adobe AcroForms, since such files probably do not need OCR. It can work with these files.
- Added three new **experimental** features to improve OCR quality in certain conditions. The name, syntax and behavior of these arguments is subject to change. They may also be incompatible with some other features.
  - `--remove-vectors` which strips out vector graphics. This can improve OCR quality since OCR will not search artwork for readable text; however, it currently removes "text as curves" as well.
  - `--mask-barcodes` to detect and suppress barcodes in files. We have observed that barcodes can interfere with OCR because they are "text-like" but not actually textual.
  - `--threshold` which uses a more sophisticated thresholding algorithm than is currently in use in Tesseract OCR. This works around a known issue in Tesseract 4.0 with dark text on bright backgrounds.
- Fixed an issue where an error message was not reported when the installed Ghostscript was very old.
- The PDF optimizer now saves files with object streams enabled when the optimization level is `--optimize 1` or higher (the default). This makes files a little bit smaller, but requires PDF 1.5. PDF 1.5 was first released in 2003 and is broadly supported by PDF viewers, but some rudimentary PDF parsers such as PyPDF2 do not understand object streams. You can use the command line tool `qpdf --object-streams=disable` or pikepdf library to remove them.
- New dependency: pdfminer.six 20181108. Note this is a fork of the Python 2-only pdfminer.
- Deprecation notice: At the end of 2018, we will be ending support for Python 3.5 and Tesseract 3.x. OCRmyPDF v7 will continue to work with older versions.

2.6 v7.2.1

- Fix compatibility with an API change in pikepdf 0.3.5.
- A kludge to support Leptonica versions older than 1.72 in the test suite was dropped. Older versions of Leptonica are likely still compatible. The only impact is that a portion of the test suite will be skipped.

2.7 v7.2.0

Lossy JBIG2 behavior change

A user reported that ocrmypdf was in fact using JBIG2 in **lossy** compression mode. This was not the intended behavior. Users should review the technical concerns with JBIG2 in lossy mode and decide if this is a concern for their use case.

JBIG2 lossy mode does achieve higher compression ratios than any other monochrome compression technology; for large text documents the savings are considerable. JBIG2 lossless still gives great compression ratios and is a major improvement over the older CCITT G4 standard.

Only users who have reviewed the concerns with JBIG2 in lossy mode should opt-in. As such, lossy mode JBIG2 is only turned on when the new argument `--jbig2-lossy` is issued. This is independent of the setting for `--optimize`.

Users who did not install an optional JBIG2 encoder are unaffected.

(Thanks to user 'bsdice' for reporting this issue.)

Other issues

- When the image optimizer quantizes an image to 1 bit per pixel, it will now attempt to further optimize that image as CCITT or JBIG2, instead of keeping it in the "flate" encoding which is not efficient for 1 bpp images. (#297)
- Images in PDFs that are used as soft masks (i.e. transparency masks or alpha channels) are now excluded from optimization.
- Fixed handling of Tesseract 4.0-rc1 which now accepts invalid Tesseract configuration files, which broke the test suite.

2.8 v7.1.0

- Improve the performance of initial text extraction, which is done to determine if a file contains existing text of some kind or not. On large files, this initial processing is now about 20x times faster. (#299)
- pikepdf 0.3.3 is now required.
- Fixed issue #231, a problem with JPEG2000 images where image metadata was only available inside the JPEG2000 file.
- Fixed some additional Ghostscript 9.25 compatibility issues.
- Improved handling of KeyboardInterrupt error messages. (#301)
- README.md is now served in GitHub markdown instead of reStructuredText.

2.9 v7.0.6

- Blacklist Ghostscript 9.24, now that 9.25 is available and fixes many regressions in 9.24.

2.10 v7.0.5

- Improve capability with Ghostscript 9.24, and enable the JPEG passthrough feature when this version in installed.
- Ghostscript 9.24 lost the ability to set PDF title, author, subject and keyword metadata to Unicode strings. OCRmyPDF will set ASCII strings and warn when Unicode is suppressed. Other software may be used to update metadata. This is a short term work around.
- PDFs generated by Kodak Capture Desktop, or generally PDFs that contain indirect references to null objects in their table of contents, would have an invalid table of contents after processing by OCRmyPDF that might interfere with other viewers. This has been fixed.
- Detect PDFs generated by Adobe LiveCycle, which can only be displayed in Adobe Acrobat and Reader currently. When these are encountered, exit with an error instead of performing OCR on the "Please wait" error message page.

2.11 v7.0.4

- Fix exception thrown when trying to optimize a certain type of PNG embedded in a PDF with the `-O2`

- Update to pikepdf 0.3.2, to gain support for optimizing some additional image types that were previously excluded from optimization (CMYK and grayscale). Fixes #285.

2.12 v7.0.3

- Fix issue #284, an error when parsing inline images that have are also image masks, by upgrading pikepdf to 0.3.1

2.13 v7.0.2

- Fix a regression with `--rotate-pages` on pages that already had rotations applied. (#279)
- Improve quality of page rotation in some cases by rasterizing a higher quality preview image. (#281)

2.14 v7.0.1

- Fix compatibility with img2pdf >= 0.3.0 by rejecting input images that have an alpha channel
- Add forward compatibility for pikepdf 0.3.0 (unrelated to img2pdf)
- Various documentation updates for v7.0.0 changes

2.15 v7.0.0

- The core algorithm for combining OCR layers with existing PDF pages has been rewritten and improved considerably. PDFs are no longer split into single page PDFs for processing; instead, images are rendered and the OCR results are grafted onto the input PDF. The new algorithm uses less temporary disk space and is much more performant especially for large files.
- New dependency: pikepdf. pikepdf is a powerful new Python PDF library driving the latest OCRmyPDF features, built on the QPDF C++ library (libqpdf).
- New feature: PDF optimization with `-O` or `--optimize`. After OCR, OCRmyPDF will perform image optimizations relevant to OCR PDFs.
  - If a JBIG2 encoder is available, then monochrome images will be converted, with the potential for huge savings on large black and white images, since JBIG2 is far more efficient than any other monochrome (bi-level) compression. (All known US patents related to JBIG2 have probably expired, but it remains the responsibility of the user to supply a JBIG2 encoder such as jbig2enc. OCRmyPDF does not implement JBIG2 encoding.)
  - If `pngquant` is installed, OCRmyPDF will optionally use it to perform lossy quantization and compression of PNG images.
  - The quality of JPEGs can also be lowered, on the assumption that a lower quality image may be suitable for storage after OCR.
  - This image optimization component will eventually be offered as an independent command line utility.
  - Optimization ranges from `-O0` through `-O3`, where `0` disables optimization and `3` implements all options. `1`, the default, performs only safe and lossless optimizations. (This is similar to GCC's optimization parameter.) The exact type of optimizations performed will vary over time.

- Small amounts of text in the margins of a page, such as watermarks, page numbers, or digital stamps, will no longer prevent the rest of a page from being OCRed when `--skip-text` is issued. This behavior is based on a heuristic.
- Removed features
  - The deprecated `--pdf-renderer tesseract` PDF renderer was removed.
  - `-g`, the option to generate debug text pages, was removed because it was a maintenance burden and only worked in isolated cases. HOCR pages can still be previewed by running the hocrtransform.py with appropriate settings.
- Removed dependencies
  - `PyPDF2`
  - `defusedxml`
  - `PyMuPDF`
- The `sandwich` PDF renderer can be used with all supported versions of Tesseract, including that those prior to v3.05 which don't support `-c textonly`. (Tesseract v4.0.0 is recommended and more efficient.)
- `--pdf-renderer auto` option and the diagnostics used to select a PDF renderer now work better with old versions, but may make different decisions than past versions.
- If everything succeeds but PDF/A conversion fails, a distinct return code is now returned (`ExitCode.pdfa_conversion_failed (10)`) where this situation previously returned `ExitCode.invalid_output_pdf (4)`. The latter is now returned only if there is some indication that the output file is invalid.
- Notes for downstream packagers
  - There is also a new dependency on `python-xmp-toolkit` which in turn depends on `libexempi3`.
  - It may be necessary to separately `pip install pycparser` to avoid another Python 3.7 issue.

2.16 v6.2.5

- Disable a failing test due to Tesseract 4.0rc1 behavior change. Previously, Tesseract would exit with an error message if its configuration was invalid, and OCRmyPDF would intercept this message. Now Tesseract issues a warning, which OCRmyPDF v6.2.5 may relay or ignore. (In v7.x, OCRmyPDF will respond to the warning.)
- This release branch no longer supports using the optional PyMuPDF installation, since it was removed in v7.x.
- This release branch no longer supports macOS. macOS users should upgrade to v7.x.

2.17 v6.2.4

- Backport Ghostscript 9.25 compatibility fixes, which removes support for setting Unicode metadata
- Backport blacklisting Ghostscript 9.24
- Older versions of Ghostscript are still supported

2.18 v6.2.3

- Fix compatibility with img2pdf >= 0.3.0 by rejecting input images that have an alpha channel
- This version will be included in Ubuntu 18.10

2.19 v6.2.2

- Backport compatibility fixes for Python 3.7 and ruffus 2.7.0 from v7.0.0
- Backport fix to ignore masks when deciding what colors are on a page
- Backport some minor improvements from v7.0.0: better argument validation and warnings about the Tesseract 4.0.0 `--user-words` regression

2.20 v6.2.1

- Fix recent versions of Tesseract (after 4.0.0-beta1) not being detected as supporting the `sandwich` renderer (#271).

2.21 v6.2.0

- **Docker**: The Docker image `ocrmypdf-tess4` has been removed. The main Docker images, `ocrmypdf` and `ocrmypdf-polyglot` now use Ubuntu 18.04 as a base image, and as such Tesseract 4.0.0-beta1 is now the Tesseract version they use. There is no Docker image based on Tesseract 3.05 anymore.
- Creation of PDF/A-3 is now supported. However, there is no ability to attach files to PDF/A-3.
- Lists more reasons why the file size might grow.
- Fix issue #262, `--remove-background` error on PDFs contained colormapped (paletted) images.
- Fix another XMP metadata validation issue, in cases where the input file's creation date has no timezone and the creation date is not overridden.

2.22 v6.1.5

- Fix issue #253, a possible division by zero when using the `hocr` renderer.
- Fix incorrectly formatted `<xmp:ModifyDate>` field inside XMP metadata for PDF/As. veraPDF flags this as a PDF/A validation failure. The error is caused the timezone and final digit of the seconds of modified time to be omitted, so at worst the modification time stamp is rounded to the nearest 10 seconds.

2.23 v6.1.4

- Fix issue #248 `--clean` argument may remove OCR from left column of text on certain documents. We now set `--layout none` to suppress this.
- The test cache was updated to reflect the change above.

- Change test suite to accommodate Ghostscript 9.23's new ability to insert JPEGs into PDFs without transcoding.
- XMP metadata in PDFs is now examined using `defusedxml` for safety.
- If an external process exits with a signal when asked to report its version, we now print the system error message instead of suppressing it. This occurred when the required executable was found but was missing a shared library.
- qpdf 7.0.0 or newer is now required as the test suite can no longer pass without it.

2.23.1 Notes

- An apparent regression in Ghostscript 9.23 will cause some ocrmypdf output files to become invalid in rare cases; the workaround for the moment is to set `--force-ocr`.

2.24 v6.1.3

- Fix issue #247, `/CreationDate` metadata not copied from input to output.
- A warning is now issued when Python 3.5 is used on files with a large page count, as this case is known to regress to single core performance. The cause of this problem is unknown.

2.25 v6.1.2

- Upgrade to PyMuPDF v1.12.5 which includes a more complete fix to #239.
- Add `defusedxml` dependency.

2.26 v6.1.1

- Fix text being reported as found on all pages if PyMuPDF is not installed.

2.27 v6.1.0

- PyMuPDF is now an optional but recommended dependency, to alleviate installation difficulties on platforms that have less access to PyMuPDF than the author anticipated. (For version 6.x only) install OCRmyPDF with `pip install ocrmypdf[fitz]` to use it to its full potential.
- Fix `FileExistsError` that could occur if OCR timed out while it was generating the output file. (#218)
- Fix table of contents/bookmarks all being redirected to page 1 when generating a PDF/A (with PyMuPDF). (Without PyMuPDF the table of contents is removed in PDF/A mode.)
- Fix "RuntimeError: invalid key in dict" when table of contents/bookmarks titles contained the character `)`. (#239)
- Added a new argument `--skip-repair` to skip the initial PDF repair step if the PDF is already well-formed (because another program repaired it).

2.28 v6.0.0

- The software license has been changed to GPLv3. Test resource files and some individual sources may have other licenses.
- OCRmyPDF now depends on PyMuPDF. Including PyMuPDF is the primary reason for the change to GPLv3.
- Other backward incompatible changes
  - The `OCRMYPDF_TESSERACT`, `OCRMYPDF_QPDF`, `OCRMYPDF_GS` and `OCRMYPDF_UNPAPER` environment variables are no longer used. Change `PATH` if you need to override the external programs OCRmyPDF uses.
  - The `ocrmypdf` package has been moved to `src/ocrmypdf` to avoid issues with accidental import.
  - The function `ocrmypdf.exec.get_program` was removed.
  - The deprecated module `ocrmypdf.pageinfo` was removed.
  - The `--pdf-renderer tess4` alias for `sandwich` was removed.
- Fixed an issue where OCRmyPDF failed to detect existing text on pages, depending on how the text and fonts were encoded within the PDF. (#233, #232)
- Fixed an issue that caused dramatic inflation of file sizes when `--skip-text --output-type pdf` was used. OCRmyPDF now removes duplicate resources such as fonts, images and other objects that it generates. (#237)
- Improved performance of the initial page splitting step. Originally this step was not believed to be expensive and ran in a process. Large file testing revealed it to be a bottleneck, so it is now parallelized. On a 700 page file with quad core machine, this change saves about 2 minutes. (#234)
- The test suite now includes a cache that can be used to speed up test runs across platforms. This also does not require computing checksums, so it's faster. (#217)

2.29 v5.7.0

- Fixed an issue that caused poor CPU utilization on machines with more than 4 cores when running Tesseract 4. (Related to issue #217.)
- The 'hocr' renderer has been improved. The 'sandwich' and 'tesseract' renderers are still better for most use cases, but 'hocr' may be useful for people who work with the PDF.js renderer in English/ASCII languages. (#225)
  - It now formats text in a matter that is easier for certain PDF viewers to select and extract copy and paste text. This should help macOS Preview and PDF.js in particular.
  - The appearance of selected text and behavior of selecting text is improved.
  - The PDF content stream now uses relative moves, making it more compact and easier for viewers to determine when two words on the same line.
  - It can now deal with text on a skewed baseline.
  - Thanks to @cforcey for the pull request, @jbreiden for many helpful suggestions, @ctbarbour for another round of improvements, and @acaloiaro for an independent review.

2.30 v5.6.3

- Suppress two debug messages that were too verbose

2.31 v5.6.2

- Development branch accidentally tagged as release. Do not use.

2.32 v5.6.1

- Fix issue #219: change how the final output file is created to avoid triggering permission errors when the output is a special file such as `/dev/null`
- Fix test suite failures due to a qpdf 8.0.0 regression and Python 3.5's handling of symlink
- The "encrypted PDF" error message was different depending on the type of PDF encryption. Now a single clear message appears for all types of PDF encryption.
- ocrmypdf is now in Homebrew. Homebrew users are advised to the version of ocrmypdf in the official homebrew-core formulas rather than the private tap.
- Some linting

2.33 v5.6.0

- Fix issue #216: preserve "text as curves" PDFs without rasterizing file
- Related to the above, messages about rasterizing are more consistent
- For consistency versions minor releases will now get the trailing .0 they always should have had.

2.34 v5.5

- Add new argument `--max-image-mpixels`. Pillow 5.0 now raises an exception when images may be decompression bombs. This argument can be used to override the limit Pillow sets.
- Fix output page cropped when using the sandwich renderer and OCR is skipped on a rotated and image-processed page
- A warning is now issued when old versions of Ghostscript are used in cases known to cause issues with non-Latin characters
- Fix a few parameter validation checks for `-output-type pdfa-1` and `pdfa-2`

2.35 v5.4.4

- Fix issue #181: fix final merge failure for PDFs with more pages than the system file handle limit (`ulimit -n`)

- Fix issue #200: an uncommon syntax for formatting decimal numbers in a PDF would cause qpdf to issue a warning, which ocrmypdf treated as an error. Now this the warning is relayed.
- Fix an issue where intermediate PDFs would be created at version 1.3 instead of the version of the original file. It's possible but unlikely this had side effects.
- A warning is now issued when older versions of qpdf are used since issues like #200 cause qpdf to infinite-loop
- Address issue #140: if Tesseract outputs invalid UTF-8, escape it and print its message instead of aborting with a Unicode error
- Adding previously unlisted setup requirement, pytest-runner
- Update documentation: fix an error in the example script for Synology with Docker images, improved security guidance, advised `pip install --user`

2.36 v5.4.3

- If a subprocess fails to report its version when queried, exit cleanly with an error instead of throwing an exception
- Added test to confirm that the system locale is Unicode-aware and fail early if it's not
- Clarified some copyright information
- Updated pinned requirements.txt so the homebrew formula captures more recent versions

2.37 v5.4.2

- Fixed a regression from v5.4.1 that caused sidecar files to be created as empty files

2.38 v5.4.1

- Add workaround for Tesseract v4.00alpha crash when trying to obtain orientation and the latest language packs are installed

2.39 v5.4

- Change wording of a deprecation warning to improve clarity
- Added option to generate PDF/A-1b output if desired (`--output-type pdfa-1`); default remains PDF/A-2b generation
- Update documentation

2.40 v5.3.3

- Fixed missing error message that should occur when trying to force `--pdf-renderer sandwich` on old versions of Tesseract
- Update copyright information in test files
- Set system `LANG` to UTF-8 in Dockerfiles to avoid UTF-8 encoding errors

2.41 v5.3.2

- Fixed a broken test case related to language packs

2.42 v5.3.1

- Fixed wrong return code given for missing Tesseract language packs
- Fixed "brew audit" crashing on Travis when trying to auto-brew

2.43 v5.3

- Added `--user-words` and `--user-patterns` arguments which are forwarded to Tesseract OCR as words and regular expressions respective to use to guide OCR. Supplying a list of subject-domain words should assist Tesseract with resolving words. (#165)
- Using a non Latin-1 language with the "hocr" renderer now warns about possible OCR quality and recommends workarounds (#176)
- Output file path added to error message when that location is not writable (#175)
- Otherwise valid PDFs with leading whitespace at the beginning of the file are now accepted

2.44 v5.2

- When using Tesseract 3.05.01 or newer, OCRmyPDF will select the "sandwich" PDF renderer by default, unless another PDF renderer is specified with the `--pdf-renderer` argument. The previous behavior was to select `--pdf-renderer=hocr`.
- The "tesseract" PDF renderer is now deprecated, since it can cause problems with Ghostscript on Tesseract 3.05.00
- The "tess4" PDF renderer has been renamed to "sandwich". "tess4" is now a deprecated alias for "sandwich".

2.45 v5.1

- Files with pages larger than 200" (5080 mm) in either dimension are now supported with `--output-type=pdf` with the page size preserved (in the PDF specification this feature is called UserUnit scaling). Due to Ghostscript limitations this is not available in conjunction with PDF/A output.

2.46 v5.0.1

- Fixed issue #169, exception due to failure to create sidecar text files on some versions of Tesseract 3.04, including the jbarlow83/ocrmypdf Docker image

2.47 v5.0

- Backward incompatible changes
  - Support for Python 3.4 dropped. Python 3.5 is now required.
  - Support for Tesseract 3.02 and 3.03 dropped. Tesseract 3.04 or newer is required. Tesseract 4.00 (alpha) is supported.
  - The OCRmyPDF.sh script was removed.
- Add a new feature, `--sidecar`, which allows creating "sidecar" text files which contain the OCR results in plain text. These OCR text is more reliable than extracting text from PDFs. Closes #126.
- New feature: `--pdfa-image-compression`, which allows overriding Ghostscript's lossy-or-lossless image encoding heuristic and making all images JPEG encoded or lossless encoded as desired. Fixes #163.
- Fixed issue #143, added `--quiet` to suppress "INFO" messages
- Fixed issue #164, a typo
- Removed the command line parameters `-n` and `--just-print` since they have not worked for some time (reported as Ubuntu bug #1687308)

2.48 v4.5.6

- Fixed issue #156, 'NoneType' object has no attribute 'getObject' on pages with no optional /Contents record. This should resolve all issues related to pages with no /Contents record.
- Fixed issue #158, ocrmypdf now stops and terminates if Ghostscript fails on an intermediate step, as it is not possible to proceed.
- Fixed issue #160, exception thrown on certain invalid arguments instead of error message

2.49 v4.5.5

- Automated update of macOS homebrew tap
- Fixed issue #154, KeyError '/Contents' when searching for text on blank pages that have no /Contents record. Note: incomplete fix for this issue.

2.50 v4.5.4

- Fix `--skip-big` raising an exception if a page contains no images (#152) (thanks to @TomRaz)
- Fix an issue where pages with no images might trigger "cannot write mode P as JPEG" (#151)

2.51 v4.5.3

- Added a workaround for Ghostscript 9.21 and probably earlier versions would fail with the error message "VMerror -25", due to a Ghostscript bug in XMP metadata handling

- High Unicode characters (U+10000 and up) are no longer accepted for setting metadata on the command line, as Ghostscript may not handle them correctly.
- Fixed an issue where the `tess4` renderer would duplicate content onto output pages if tesseract failed or timed out
- Fixed `tess4` renderer not recognized when lossless reconstruction is possible

2.52 v4.5.2

- Fix issue #147. `--pdf-renderer tess4 --clean` will produce an oversized page containing the original image in the bottom left corner, due to loss DPI information.
- Make "using Tesseract 4.0" warning less ominous
- Set up machinery for homebrew OCRmyPDF tap

2.53 v4.5.1

- Fix issue #137, proportions of images with a non-square pixel aspect ratio would be distorted in output for `--force-ocr` and some other combinations of flags

2.54 v4.5

- PDFs containing "Form XObjects" are now supported (issue #134; PDF reference manual 8.10), and images they contain are taken into account when determining the resolution for rasterizing
- The Tesseract 4 Docker image no longer includes all languages, because it took so long to build something would tend to fail
- OCRmyPDF now warns about using `--pdf-renderer tesseract` with Tesseract 3.04 or lower due to issues with Ghostscript corrupting the OCR text in these cases

2.55 v4.4.2

- The Docker images (ocrmypdf, ocrmypdf-polyglot, ocrmypdf-tess4) are now based on Ubuntu 16.10 instead of Debian stretch
  - This makes supporting the Tesseract 4 image easier
  - This could be a disruptive change for any Docker users who built customized these images with their own changes, and made those changes in a way that depends on Debian and not Ubuntu
- OCRmyPDF now prevents running the Tesseract 4 renderer with Tesseract 3.04, which was permitted in v4.4 and v4.4.1 but will not work

2.56 v4.4.1

- To prevent a TIFF output error caused by img2pdf >= 0.2.1 and Pillow <= 3.4.2, dependencies have been tightened

- The Tesseract 4.00 simultaneous process limit was increased from 1 to 2, since it was observed that 1 lowers performance
- Documentation improvements to describe the `--tesseract-config` feature
- Added test cases and fixed error handling for `--tesseract-config`
- Tweaks to setup.py to deal with issues in the v4.4 release

2.57 v4.4

- Tesseract 4.00 is now supported on an experimental basis.
  - A new rendering option `--pdf-renderer tess4` exploits Tesseract 4's new text-only output PDF mode. See the documentation on PDF Renderers for details.
  - The `--tesseract-oem` argument allows control over the Tesseract 4 OCR engine mode (tesseract's `--oem`). Use `--tesseract-oem 2` to enforce the new LSTM mode.
  - Fixed poor performance with Tesseract 4.00 on Linux
- Fixed an issue that caused corruption of output to stdout in some cases
- Removed test for Pillow JPEG and PNG support, as the minimum supported version of Pillow now enforces this
- OCRmyPDF now tests that the intended destination file is writable before proceeding
- The test suite now requires `pytest-helpers-namespace` to run (but not install)
- Significant code reorganization to make OCRmyPDF re-entrant and improve performance. All changes should be backward compatible for the v4.x series.
  - However, OCRmyPDF's dependency "ruffus" is not re-entrant, so no Python API is available. Scripts should continue to use the command line interface.

2.58 v4.3.5

- Update documentation to confirm Python 3.6.0 compatibility. No code changes were needed, so many earlier versions are likely supported.

2.59 v4.3.4

- Fixed "decimal.InvalidOperation: quantize result has too many digits" for high DPI images

2.60 v4.3.3

- Fixed PDF/A creation with Ghostscript 9.20 properly
- Fixed an exception on inline stencil masks with a missing optional parameter

2.61 v4.3.2

- Fixed a PDF/A creation issue with Ghostscript 9.20 (note: this fix did not actually work)

2.62 v4.3.1

- Fixed an issue where pages produced by the "hocr" renderer after a Tesseract timeout would be rotated incorrectly if the input page was rotated with a /Rotate marker
- Fixed a file handle leak in LeptonicaErrorTrap that would cause a "too many open files" error for files around hundred pages of pages long when `--deskew` or `--remove-background` or other Leptonica based image processing features were in use, depending on the system value of `ulimit -n`
- Ability to specify multiple languages for multilingual documents is now advertised in documentation
- Reduced the file sizes of some test resources
- Cleaned up debug output
- Tesseract caching in test cases is now more cautious about false cache hits and reproducing exact output, not that any problems were observed

2.63 v4.3

- New feature `--remove-background` to detect and erase the background of color and grayscale images
- Better documentation
- Fixed an issue with PDFs that draw images when the raster stack depth is zero
- ocrmypdf can now redirect its output to stdout for use in a shell pipeline
  - This does not improve performance since temporary files are still used for buffering
  - Some output validation is disabled in this mode

2.64 v4.2.5

- Fixed an issue (#100) with PDFs that omit the optional /BitsPerComponent parameter on images
- Removed non-free file milk.pdf

2.65 v4.2.4

- Fixed an error (#90) caused by PDFs that use stencil masks properly
- Fixed handling of PDFs that try to draw images or stencil masks without properly setting up the graphics state (such images are now ignored for the purposes of calculating DPI)

2.66 v4.2.3

- Fixed an issue with PDFs that store page rotation (/Rotate) in an indirect object
- Integrated a few fixes to simplify downstream packaging (Debian)
  - The test suite no longer assumes it is installed
  - If running Linux, skip a test that passes Unicode on the command line
- Added a test case to check explicit masks and stencil masks
- Added a test case for indirect objects and linearized PDFs
- Deprecated the OCRmyPDF.sh shell script

2.67 v4.2.2

- Improvements to documentation

2.68 v4.2.1

- Fixed an issue where PDF pages that contained stencil masks would report an incorrect DPI and cause Ghostscript to abort
- Implemented stdin streaming

2.69 v4.2

- ocrmypdf will now try to convert single image files to PDFs if they are provided as input (#15)
  - This is a basic convenience feature. It only supports a single image and always makes the image fill the whole page.
  - For better control over image to PDF conversion, use `img2pdf` (one of ocrmypdf's dependencies)
- New argument `--output-type {pdf|pdfa}` allows disabling Ghostscript PDF/A generation
  - `pdfa` is the default, consistent with past behavior
  - `pdf` provides a workaround for users concerned about the increase in file size from Ghostscript forcing JBIG2 images to CCITT and transcoding JPEGs
  - `pdf` preserves as much as it can about the original file, including problems that PDF/A conversion fixes
- PDFs containing images with "non-square" pixel aspect ratios, such as 200x100 DPI, are now handled and converted properly (fixing a bug that caused to be cropped)
- `--force-ocr` rasterizes pages even if they contain no images
  - supports users who want to use OCRmyPDF to reconstruct text information in PDFs with damaged Unicode maps (copy and paste text does not match displayed text)
  - supports reinterpreting PDFs where text was rendered as curves for printing, and text needs to be recovered
  - fixes issue #82

- Fixes an issue where, with certain settings, monochrome images in PDFs would be converted to 8-bit grayscale, increasing file size (#79)
- Support for Ubuntu 12.04 LTS "precise" has been dropped in favor of (roughly) Ubuntu 14.04 LTS "trusty"
  - Some Ubuntu "PPAs" (backports) are needed to make it work
- Support for some older dependencies dropped
  - Ghostscript 9.15 or later is now required (available in Ubuntu trusty with backports)
  - Tesseract 3.03 or later is now required (available in Ubuntu trusty)
- Ghostscript now runs in "safer" mode where possible

2.70 v4.1.4

- Bug fix: monochrome images with an ICC profile attached were incorrectly converted to full color images if lossless reconstruction was not possible due to other settings; consequence was increased file size for these images

2.71 v4.1.3

- More helpful error message for PDFs with version 4 security handler
- Update usage instructions for Windows/Docker users
- Fix order of operations for matrix multiplication (no effect on most users)
- Add a few leptonica wrapper functions (no effect on most users)

2.72 v4.1.2

- Replace IEC sRGB ICC profile with Debian's sRGB (from icc-profiles-free) which is more compatible with the MIT license
- More helpful error message for an error related to certain types of malformed PDFs

2.73 v4.1

- `--rotate-pages` now only rotates pages when reasonably confidence in the orientation. This behavior can be adjusted with the new argument `--rotate-pages-threshold`
- Fixed problems in error checking if `unpaper` is uninstalled or missing at run-time
- Fixed problems with "RethrownJobError" errors during error handling that suppressed the useful error messages

2.74 v4.0.7

- Minor correction to Ghostscript output settings

2.75 v4.0.6

- Update install instructions
- Provide a sRGB profile instead of using Ghostscript's

2.76 v4.0.5

- Remove some verbose debug messages from v4.0.4
- Fixed temporary that wasn't being deleted
- DPI is now calculated correctly for cropped images, along with other image transformations
- Inline images are now checked during DPI calculation instead of rejecting the image

2.77 v4.0.4

Released with verbose debug message turned on. Do not use. Skip to v4.0.5.

2.78 v4.0.3

New features

- Page orientations detected are now reported in a summary comment

Fixes

- Show stack trace if unexpected errors occur
- Treat "too few characters" error message from Tesseract as a reason to skip that page rather than abort the file
- Docker: fix blank JPEG2000 issue by insisting on Ghostscript versions that have this fixed

2.79 v4.0.2

Fixes

- Fixed compatibility with Tesseract 3.04.01 release, particularly its different way of outputting orientation information
- Improved handling of Tesseract errors and crashes
- Fixed use of chmod on Docker that broke most test cases

2.80 v4.0.1

Fixes

- Fixed a KeyError if tesseract fails to find page orientation information

2.81 v4.0

New features

- Automatic page rotation (`-r`) is now available. It uses ignores any prior rotation information on PDFs and sets rotation based on the dominant orientation of detectable text. This feature is fairly reliable but some false positives occur especially if there is not much text to work with. (#4)
- Deskewing is now performed using Leptonica instead of unpaper. Leptonica is faster and more reliable at image deskewing than unpaper.

Fixes

- Fixed an issue where lossless reconstruction could cause some pages to be appear incorrectly if the page was rotated by the user in Acrobat after being scanned (specifically if it a /Rotate tag)
- Fixed an issue where lossless reconstruction could misalign the graphics layer with respect to text layer if the page had been cropped such that its origin is not (0, 0) (#49)

Changes

- Logging output is now much easier to read
- `--deskew` is now performed by Leptonica instead of unpaper (#25)
- libffi is now required
- Some changes were made to the Docker and Travis build environments to support libffi
- `--pdf-renderer=tesseract` now displays a warning if the Tesseract version is less than 3.04.01, the planned release that will include fixes to an important OCR text rendering bug in Tesseract 3.04.00. You can also manually install ./share/sharp2.ttf on top of pdf.ttf in your Tesseract tessdata folder to correct the problem.

2.82 v3.2.1

Changes

- Fixed issue #47 "convert() got and unexpected keyword argument 'dpi'" by upgrading to img2pdf 0.2
- Tweaked the Dockerfiles

2.83 v3.2

New features

- Lossless reconstruction: when possible, OCRmyPDF will inject text layers without otherwise manipulating the content and layout of a PDF page. For example, a PDF containing a mix of vector and raster content would see the vector content preserved. Images may still be transcoded during PDF/A conversion. (`--deskew` and `--clean-final` disable this mode, necessarily.)
- New argument `--tesseract-pagesegmode` allows you to pass page segmentation arguments to Tesseract OCR. This helps for two column text and other situations that confuse Tesseract.
- Added a new "polyglot" version of the Docker image, that generates Tesseract with all languages packs installed, for the polyglots among us. It is much larger.

Changes

- JPEG transcoding quality is now 95 instead of the default 75. Bigger file sizes for less degradation.

2.84 v3.1.1

Changes

- Fixed bug that caused incorrect page size and DPI calculations on documents with mixed page sizes

2.85 v3.1

Changes

- Default output format is now PDF/A-2b instead of PDF/A-1b
- Python 3.5 and macOS El Capitan are now supported platforms - no changes were needed to implement support
- Improved some error messages related to missing input files
- Fixed issue #20 - uppercase .PDF extension not accepted
- Fixed an issue where OCRmyPDF failed to text that certain pages contained previously OCR'ed text, such as OCR text produced by Tesseract 3.04
- Inserts /Creator tag into PDFs so that errors can be traced back to this project
- Added new option `--pdf-renderer=auto`, to let OCRmyPDF pick the best PDF renderer. Currently it always chooses the 'hocrtransform' renderer but that behavior may change.
- Set up Travis CI automatic integration testing

2.86 v3.0

New features

- Easier installation with a Docker container or Python's `pip` package manager
- Eliminated many external dependencies, so it's easier to setup
- Now installs `ocrmypdf` to `/usr/local/bin` or equivalent for system-wide access and easier typing
- Improved command line syntax and usage help (`--help`)
- Tesseract 3.03+ PDF page rendering can be used instead for better positioning of recognized text (`--pdf-renderer tesseract`)
- PDF metadata (title, author, keywords) are now transferred to the output PDF
- PDF metadata can also be set from the command line (`--title`, etc.)
- Automatic repairs malformed input PDFs if possible
- Added test cases to confirm everything is working
- Added option to skip extremely large pages that take too long to OCR and are often not OCRable (e.g. large scanned maps or diagrams); other pages are still processed (`--skip-big`)
- Added option to kill Tesseract OCR process if it seems to be taking too long on a page, while still processing other pages (`--tesseract-timeout`)
- Less common colorspaces (CMYK, palette) are now supported by conversion to RGB
- Multiple images on the same PDF page are now supported

Changes

- New, robust rewrite in Python 3.4+ with ruffus pipelines
- Now uses Ghostscript 9.14's improved color conversion model to preserve PDF colors
- OCR text is now rendered in the PDF as invisible text. Previous versions of OCRmyPDF incorrectly rendered visible text with an image on top.
- All "tasks" in the pipeline can be executed in parallel on any available CPUs, increasing performance
- The `-o DPI` argument has been phased out, in favor of `--oversample DPI`, in case we need `-o OUTPUTFILE` in the future
- Removed several dependencies, so it's easier to install. We no longer use:
  - GNU parallel
  - ImageMagick
  - Python 2.7
  - Poppler
  - MuPDF tools
  - shell scripts
  - Java and JHOVE
  - libxml2
- Some new external dependencies are required or optional, compared to v2.x:
  - Ghostscript 9.14+
  - qpdf 5.0.0+
  - Unpaper 6.1 (optional)
  - some automatically managed Python packages

Release candidates^

- rc9:
  - fix issue #118: report error if ghostscript iccprofiles are missing
  - fixed another issue related to #111: PDF rasterized to palette file
  - add support image files with a palette
  - don't try to validate PDF file after an exception occurs
- rc8:
  - fix issue #111: exception thrown if PDF is missing DocumentInfo dictionary
- rc7:
  - fix error when installing direct from pip, "no such file 'requirements.txt'"
- rc6:
  - dropped libxml2 (Python lxml) since Python 3's internal XML parser is sufficient
  - set up Docker container
  - fix Unicode errors if recognized text contains Unicode characters and system locale is not UTF-8
- rc5:

  - dropped Java and JHOVE in favour of qpdf
  - improved command line error output
  - additional tests and bug fixes
  - tested on Ubuntu 14.04 LTS
- rc4:
  - dropped MuPDF in favour of qpdf
  - fixed some installer issues and errors in installation instructions
  - improve performance: run Ghostscript with multithreaded rendering
  - improve performance: use multiple cores by default
  - bug fix: checking for wrong exception on process timeout
- rc3: skipping version number intentionally to avoid confusion with Tesseract
- rc2: first release for public testing to test-PyPI, Github
- rc1: testing release process

2.87 Compatibility notes

- `./OCRmyPDF.sh` script is still available for now
- Stacking the verbosity option like `-vvv` is no longer supported
- The configuration file `config.sh` has been removed. Instead, you can feed a file to the arguments for common settings:

```
ocrmypdf input.pdf output.pdf @settings.txt
```

where `settings.txt` contains *one argument per line*, for example:

```
-l
deu
--author
A. Merkel
--pdf-renderer
tesseract
```

Fixes

- Handling of filenames containing spaces: fixed

Notes and known issues

- Some dependencies may work with lower versions than tested, so try overriding dependencies if they are "in the way" to see if they work.
- `--pdf-renderer tesseract` will output files with an incorrect page size in Tesseract 3.03, due to a bug in Tesseract.
- PDF files containing "inline images" are not supported and won't be for the 3.0 release. Scanned images almost never contain inline images.

2.88 v2.2-stable (2014-09-29)

OCRmyPDF versions 1 and 2 were implemented as shell scripts. OCRmyPDF 3.0+ is a fork that gradually replaced all shell scripts with Python while maintaining the existing command line arguments. No one is maintaining old versions.

For details on older versions, see the final version of its release notes.

CHAPTER 3

Installation

The easiest way to install OCRmyPDF is to follow the steps for your operating system/platform, although sometimes this version may be out of date.

If you want to use the latest version of OCRmyPDF, your best bet is to install the most recent version your platform provides, and then upgrade that version by installing the Python binary wheels.

Platform-specific steps

3.1 Installing on Linux

3.1.1 Debian and Ubuntu 16.10 or newer

OCRmyPDF versions in Debian & Ubuntu

Users of Debian 9 ("stretch") or later or Ubuntu 16.10 or later may simply

```
apt-get install ocrmypdf
```

As indicated in the table above, Debian and Ubuntu releases may lag behind the latest version. If the version available for your platform is out of date, you could opt to install the latest version from source. See *Installing HEAD revision from sources*.

For full details on version availability for your platform, check the Debian Package Tracker or Ubuntu launchpad.net.

Note: OCRmyPDF for Debian and Ubuntu currently omit the JBIG2 encoder. OCRmyPDF works fine without it but will produce larger output files. If you build jbig2enc from source, ocrmypdf 7.0.0 and later will automatically detect it (specifically the `jbig2` binary) on the `PATH`. To add JBIG2 encoding, see *Installing the JBIG2 encoder*.

3.1.2 Fedora 29 or newer

OCRmyPDF version

Users of Fedora 29 later may simply

```
dnf install ocrmypdf
```

For full details on version availability, check the Fedora Package Tracker.

If the version available for your platform is out of date, you could opt to install the latest version from source. See *Installing HEAD revision from sources*.

Note: OCRmyPDF for Fedora currently omits the JBIG2 encoder due to patent issues. OCRmyPDF works fine without it but will produce larger output files. If you build jbig2enc from source, ocrmypdf 7.0.0 and later will automatically detect it on the `PATH`. To add JBIG2 encoding, see **'Installing the JBIG2 encoder'_**.

3.1.3 Installing the latest version on Ubuntu 18.04 LTS

Ubuntu 18.04 includes ocrmypdf 6.1.2. To install a more recent version, first install the system version to get most of the dependencies:

```
sudo apt-get update
sudo apt-get install \
    ocrmypdf \
    python3-pip
```

There are a few dependency changes between ocrmypdf 6.1.2 and 7.x. Let's get these, too.

```
sudo apt-get install \
    libexempi3 \
    pngquant
```

Then install the most recent ocrmypdf for the local user and set the user's `PATH` to check for the user's Python packages.

```
export PATH=$HOME/.local/bin:$PATH
pip3 install --user ocrmypdf
```

To add JBIG2 encoding, see *Installing the JBIG2 encoder*.

3.1.4 Ubuntu 16.04 LTS

No package is available for Ubuntu 16.04. OCRmyPDF 8.0 and newer require Python 3.6. Ubuntu 16.04 ships Python 3.5, but you can install Python 3.6 on it. Or, you can skip Python 3.6 and install OCRmyPDF 7.x or older - for that procedure, please see the installation documentation for the version of OCRmyPDF you plan to use.

Install system packages for OCRmyPDF

```
sudo apt-get update
sudo apt-get install -y software-properties-common python-software-properties
sudo add-apt-repository -y \
    ppa:jonathonf/python-3.6 \
    ppa:alex-p/tesseract-ocr
sudo apt-get update
sudo apt-get install -y \
    ghostscript \
    libexempi3 \
    libffi6 \
    pngquant \
    python3.6 \
    qpdf \
    tesseract-ocr \
    unpaper
```

This will install a Python 3.6 binary at `/usr/bin/python3.6` alongside the system's Python 3.5. Do not remove the system Python. This will also install Tesseract 4.0 from a PPA, since the version available in Ubuntu 16.04 is too old for OCRmyPDF.

Now install pip for Python 3.6. This will install the Python 3.6 version of `pip` at `/usr/local/bin/pip`.

```
curl https://bootstrap.pypa.io/get-pip.py | sudo python3.6
```

Install OCRmyPDF

OCRmyPDF requires the locale to be set for UTF-8. **On some minimal Ubuntu installations systems**, it may be necessary to set the locale.

```
# Optional: Only need to set these if they are not already set
export LC_ALL=C.UTF-8
export LANG=C.UTF-8
```

Now install OCRmyPDF for the current user, and ensure that the `PATH` environment variable contains `$HOME/.local/bin`.

```
export PATH=$HOME/.local/bin:$PATH
pip3 install --user ocrmypdf
```

To add JBIG2 encoding, see *Installing the JBIG2 encoder*.

3.1.5 Ubuntu 14.04 LTS

Installing on Ubuntu 14.04 LTS (trusty) is more difficult than some other options, because of its age. Several backports are required. For explanations of some steps of this procedure, see the similar steps for Ubuntu 16.04.

Install system dependencies:

```
sudo apt-get update
sudo apt-get install \
    software-properties-common python-software-properties \
    zlib1g-dev \
    libexempi3 \
    libjpeg-dev \
    libffi-dev \
    pngquant \
    qpdf
```

We will need backports of Ghostscript 9.16, libav-11 (for unpaper 6.1), Tesseract 4.00 (alpha), and Python 3.6. This will replace Ghostscript and Tesseract 3.x on your system. Python 3.6 will be installed alongside the system Python 3.4.

If you prefer to not modify your system in this matter, consider using a Docker container.

```
sudo add-apt-repository ppa:vshn/ghostscript -y
sudo add-apt-repository ppa:heyarje/libav-11 -y
sudo add-apt-repository ppa:alex-p/tesseract-ocr -y
sudo add-apt-repository ppa:jonathonf/python-3.6 -y

sudo apt-get update

sudo apt-get install \
    python3.6-dev \
    ghostscript \
    tesseract-ocr \
    tesseract-ocr-eng \
    libavformat56 libavcodec56 libavutil54 \
    wget
```

Now we need to install `pip` and let it install ocrmypdf:

```
curl https://bootstrap.pypa.io/ez_setup.py -o - | python3.6 && python3.6 -m easy_
↪install pip
pip3.6 install ocrmypdf
```

These installation instructions omit the optional dependency `unpaper`, which is only available at version 0.4.2 in Ubuntu 14.04. The author could not find a backport of `unpaper`, and created a .deb package to do the job of installing unpaper 6.1 (for x86 64-bit only):

```
wget -q 'https://www.dropbox.com/s/vaq0kbwi6e6au80/unpaper_6.1-1.deb?raw=1' -O␣
↪unpaper_6.1-1.deb
sudo dpkg -i unpaper_6.1-1.deb
```

To add JBIG2 encoding, see *Installing the JBIG2 encoder*.

3.1.6 ArchLinux

The author is aware of an ArchLinux User Repository package for ocrmypdf. You can use the following command.

```
yaourt -S ocrmypdf
```

If you have any difficulties with installation, check the repository package page.

3.1.7 Other Linux packages

See the Repology page.

In general, first install the OCRmyPDF package for your system, then optionally use the procedure *Installing with Python pip* to install a more recent version.

3.2 Installing on macOS

3.2.1 Homebrew

OCRmyPDF is now a standard Homebrew formula. To install on macOS:

```
brew install ocrmypdf
```

Note: Users who previously installed OCRmyPDF on macOS using `pip install ocrmypdf` should remove the pip version (`pip3 uninstall ocrmypdf`) before switching to the Homebrew version.

Note: Users who previously installed OCRmyPDF from the private tap should switch to the mainline version (`brew untap jbarlow83/ocrmypdf`) and install from there.

3.2.2 Manual installation on macOS

These instructions probably work on all macOS supported by Homebrew.

If it's not already present, install Homebrew.

Update Homebrew:

```
brew update
```

Install or upgrade the required Homebrew packages, if any are missing. To do this, download the `Brewfile` that lists all of the dependencies to the current directory, and run `brew bundle` to process them (installing or upgrading as needed). `Brewfile` is a plain text file.

```
wget https://github.com/jbarlow83/OCRmyPDF/raw/master/.travis/Brewfile
brew bundle
```

This will include the English, French, German and Spanish language packs. If you need other languages you can optionally install them all:

```
brew install tesseract --with-all-languages  # Option 2: for all language packs
```

Update the homebrew pip:

```
pip3 install --upgrade pip
```

You can then install OCRmyPDF from PyPI, for the current user:

```
pip3 install --user ocrmypdf
```

or system-wide:

```
pip3 install ocrmypdf
```

The command line program should now be available:

```
ocrmypdf --help
```

3.3 Installing the Docker image

For some users, installing the Docker image will be easier than installing all of OCRmyPDF's dependencies. For Windows, it is the only option.

If you have Docker installed on your system, you can install a Docker image of the latest release.

Follow the Docker installation instructions for your platform. If you can run this command successfully, your system is ready to download and execute the image:

```
docker run hello-world
```

OCRmyPDF will use all available CPU cores. By default, the VirtualBox machine instance on Windows and macOS has only a single CPU core enabled. Use the VirtualBox Manager to determine the name of your Docker engine host, and then follow these optional steps to enable multiple CPUs:

```
# Optional step for Mac OS X users
docker-machine stop "yourVM"
VBoxManage modifyvm "yourVM" --cpus 2  # or whatever number of core is desired
docker-machine start "yourVM"
eval $(docker-machine env "yourVM")
```

Assuming you have a Docker engine running, you can download one of the three available images:

Image name	Download command	Notes
ocrmypdf	`docker pull jbarlow83/ocrmypdf`	Latest ocrmypdf with Tesseract 4.0.0-beta1 on Ubuntu 18.04. Includes English, French, German, Spanish, Portugeuse and Simplified Chinese.
ocrmypdf-polyglot	`docker pull jbarlow83/ocrmypdf-polyglot`	As above, with all available language packs.
ocrmypdf-webservice	`docker pull jbarlow83/ocrmypdf-polyglot`	All language packs, and a simple HTTP wrapper allowing OCRmyPDF to be used as a web service. Note that this component is licensed under AGPLv3.

For example:

```
docker pull jbarlow83/ocrmypdf
```

Then tag it to give a more convenient name, just ocrmypdf:

```
docker tag jbarlow83/ocrmypdf ocrmypdf
```

The alternative "polyglot" image provides all available language packs.

You can then run ocrmypdf using the command:

```
docker run --rm ocrmypdf --help
```

To execute the OCRmyPDF on a local file, you must provide a writable volume to the Docker image, and both the input and output file must be inside the writable volume. This example command uses the current working directory as the writable volume:

```
docker run --rm -v "$(pwd):/home/docker" <other docker arguments>   ocrmypdf <your␣
↪arguments to ocrmypdf>
```

In this worked example, the current working directory contains an input file called `test.pdf` and the output will go to `output.pdf`:

```
docker run --rm -v "$(pwd):/home/docker"   ocrmypdf --skip-text test.pdf output.pdf
```

Note: The working directory should be a writable local volume or Docker may not have permission to access it.

Note that `ocrmypdf` has its own separate `-v VERBOSITYLEVEL` argument to control debug verbosity. All Docker arguments should before the `ocrmypdf` image name and all arguments to `ocrmypdf` should be listed after.

In some environments the permissions associated with Docker can be complex to configure. The process that executes Docker may end up not having the permissions to write the specified file system. In that case one can stream the file into and out of the Docker process and avoid all permission hassles, using - as the input and output filename:

```
docker run --rm -i   ocrmypdf <other arguments to ocrmypdf> - - <input.pdf >output.pdf
```

For convenience, a shell alias can hide the docker command:

```
alias ocrmypdf='docker run --rm -v "$(pwd):/home/docker" ocrmypdf'
ocrmypdf --version  # runs docker version
```

Or in the wonderful fish shell:

```
alias ocrmypdf 'docker run --rm -v (pwd):/home/docker ocrmypdf'
funcsave ocrmypdf
```

Note: The ocrmypdf Docker images are designed for application delivery, to enable use of OCRmyPDF without fussing with dependencies. `docker run --rm` argument tells Docker to delete the container after it runs, because each container is only good for a single job. The Docker image is not designed for use as a persistent web service or for use on Amazon EC2 Container Service (AWS ECS).

3.4 Installing on Windows

Direct installation on Windows is not possible. *Install the Docker* container as described above. Ensure that your command prompt can run the docker "hello world" container.

It would probably not be too difficult to run on Windows. The main reason this has been avoided is the difficulty of packaging and installing the various non-Python dependencies: Tesseract, QPDF, Ghostscript, Leptonica. Pull requests to add or improve Windows support would be quite welcome.

The command line syntax to run ocrmypdf from a command prompt will resemble:

```
docker run -v /c/Users/sampleuser:/home/docker ocrmypdf --skip-text test.pdf output.
↪pdf
```

where /c/Users/sampleuser is a Unix representation of the Windows path C:\Users\sampleuser, assuming a user named "sampleuser" is running ocrmypdf on a file in their home directory, and the files "test.pdf" and "output.pdf" are in the sampleuser folder. The Windows user must have read and write permissions.

Bash on Ubuntu on Windows should also be a viable route for running the OCRmyPDF Docker container.

3.5 Installing with Python pip

OCRmyPDF is delivered by PyPI because it is a convenient way to install the latest version. However, PyPI and `pip` cannot address the fact that `ocrmypdf` depends on certain non-Python system libraries and programs being instsalled.

For best results, first install your platform's version of `ocrmypdf`, using the instructions elsewhere in this document. Then you can use `pip` to get the latest version if your platform version is out of date. Chances are that this will satisfy most dependencies.

Use `ocrmypdf --version` to confirm what version was installed.

Then you can install the latest OCRmyPDF from the Python wheels. First try:

```
pip3 install --user ocrmypdf
```

You should then be able to run `ocrmypdf --version` and see that the latest version was located.

Since `pip3 install --user` does not work correctly on some platforms, notably Ubuntu 16.04 and older, and the Homebrew version of Python, instead use this for a system wide installation:

```
pip3 install ocrmypdf
```

OCRmyPDF currently requires these external programs and libraries to be installed, and must be satisfied using the operating system package manager. `pip` cannot provide them.

- Python 3.6 or newer
- Ghostscript 9.15 or newer
- qpdf 8.1.0 or newer
- Tesseract 4.0.0-alpha or newer

As of ocrmypdf 7.2.1, the following versions are recommended:

- Python 3.7
- Ghostscript 9.23 or newer
- qpdf 8.2.1
- Tesseract 4.0.0 or newer
- jbig2enc 0.29 or newer
- pngquant 2.5 or newer
- unpaper 6.1

jbig2enc, pngquant, and unpaper are optional. If missing certain features are disabled. OCRmyPDF will discover them as soon as they are available.

jbig2enc, if present, will be used to optimize the encoding of monochrome images. This can significantly reduce the file size of the output file. It is not required. jbig2enc is not generally available for Ubuntu or Debian due to lingering concerns about patent issues, but can easily be built from source. To add JBIG2 encoding, see *Installing the JBIG2 encoder*.

pngquant, if present, is optionally used to optimize the encoding of PNG-style images in PDFs (actually, any that are that losslessly encoded) by lossily quantizing to a smaller color palette. It is only activated then the `--optimize` argument is `2` or `3`.

unpaper, if present, enables the `--clean` and `--clean-final` command line options.

These are in addition to the Python packaging dependencies, meaning that unfortunately, the `pip install` command cannot satisfy all of them.

3.6 Installing HEAD revision from sources

If you have `git` and Python 3.6 or newer installed, you can install from source. When the `pip` installer runs, it will alert you if dependencies are missing.

If you prefer to build every from source, you will need to build pikepdf from source. First ensure you can build and install pikepdf.

To install the HEAD revision from sources in the current Python 3 environment:

```
pip3 install git+https://github.com/jbarlow83/OCRmyPDF.git
```

Or, to install in development mode, allowing customization of OCRmyPDF, use the `-e` flag:

```
pip3 install -e git+https://github.com/jbarlow83/OCRmyPDF.git
```

You may find it easiest to install in a virtual environment, rather than system-wide:

```
git clone -b master https://github.com/jbarlow83/OCRmyPDF.git
python3 -m venv
source venv/bin/activate
cd OCRmyPDF
pip3 install .
```

However, `ocrmypdf` will only be accessible on the system PATH when you activate the virtual environment.

To run the program:

```
ocrmypdf --help
```

If not yet installed, the script will notify you about dependencies that need to be installed. The script requires specific versions of the dependencies. Older version than the ones mentioned in the release notes are likely not to be compatible to OCRmyPDF.

To install all of the development and test requirements:

```
git clone -b master https://github.com/jbarlow83/OCRmyPDF.git
python3 -m venv
source venv/bin/activate
cd OCRmyPDF
pip install -e .
pip install -r requirements/dev.txt -r requirements/test.txt
```

To add JBIG2 encoding, see *Installing the JBIG2 encoder*.

CHAPTER 4

Installing additional language packs

OCRmyPDF uses Tesseract for OCR, and relies on its language packs for languages other than English.

Tesseract supports most languages.

For Linux users, you can often find packages that provide language packs:

4.1 Debian and Ubuntu users

```
# Display a list of all Tesseract language packs
apt-cache search tesseract-ocr

# Install Chinese Simplified language pack
apt-get install tesseract-ocr-chi-sim
```

You can then pass the `-l LANG` argument to OCRmyPDF to give a hint as to what languages it should search for. Multiple languages can be requested using either `-l eng+fre` (English and French) or `-l eng -l fre`.

4.2 Fedora users

```
# Display a list of all Tesseract language packs
dnf search tesseract

# Install Chinese Simplified language pack
dnf install tesseract-langpack-chi_sim
```

You can then pass the `-l LANG` argument to OCRmyPDF to give a hint as to what languages it should search for. Multiple languages can be requested using either `-l eng+fre` (English and French) or `-l eng -l fre`.

4.3 macOS users

You can install additional language packs by *installing Tesseract using Homebrew with all language packs*.

4.4 Docker users

Users of the Docker image may use the alternative *"polyglot" container* which includes all languages.

4.4.1 Adding individual language packs to a Docker image

If you wish to add a single language pack, you could do the following:

- Download the desired `.traineddata` file from the tessdata repository. Let's use Hebrew in this example (`heb.traineddata`)
- Copy the file to `/home/user/downloads/heb.traineddata`.
- Create a new container based on the ocrmypdf-tess4 image and jump into it with a terminal:

```
host$ docker run  -v /home/user/downloads:/home/docker -it --entrypoint /bin/bash␣
↪ocrmypdf-tess4
```

- Put the file where Tesseract expects it:

```
docker$ cp /home/docker/heb.traineddata /usr/share/tesseract-ocr/tessdata
```

- Note the container id, and save it as a new image (in this example, `ocrmypdf-tess4-heb`)

```
host$ docker commit <container_id> ocrmypdf-tess4-heb
```

CHAPTER 5

Installing the JBIG2 encoder

Most Linux distributions do not include a JBIG2 encoder since JBIG2 encoding was patented for a long time. All known JBIG2 US patents have expired as of 2017, but it is possible that unknown patents exist.

JBIG2 encoding is recommended for OCRmyPDF and is used to losslessly create smaller PDFs. If JBIG2 encoding not available, lower quality encodings will be used.

JBIG2 decoding is not patented and is performed automatically by most PDF viewers. It is widely supported has been part of the PDF specification since 2001.

On macOS, Homebrew packages jbig2enc and OCRmyPDF includes it by default. The Docker image for OCRmyPDF also builds its own JBIG2 encoder from source.

For all other Linux, you must build a JBIG2 encoder from source:

```
git clone https://github.com/agl/jbig2enc
cd jbig2enc
./autogen.sh
./configure && make
[sudo] make install
```

5.1 Lossy mode JBIG2

OCRmyPDF provides lossy mode JBIG2 as an advanced feature. Users should review the technical concerns with JBIG2 in lossy mode and decide if this feature is acceptable for their use case.

JBIG2 lossy mode does achieve higher compression ratios than any other monochrome (bitonal) compression technology; for large text documents the savings are considerable. JBIG2 lossless still gives great compression ratios and is a major improvement over the older CCITT G4 standard. As explained above, there is some risk of substitution errors.

To turn on JBIG2 lossy mode, add the argument `--jbig2-lossy`. `--optimize {1,2,3}` are necessary for the argument to take effect also required. Also, a JBIG2 encoder must be installed as described in the previous section.

ocrmypdf v7.0 and v7.1 used lossy mode by default.

CHAPTER 6

Cookbook

6.1 Basic examples

6.1.1 Help!

ocrmypdf has built-in help.

```
ocrmypdf --help
```

6.1.2 Add an OCR layer and convert to PDF/A

```
ocrmypdf input.pdf output.pdf
```

6.1.3 Add an OCR layer and output a standard PDF

```
ocrmypdf --output-type pdf input.pdf output.pdf
```

6.1.4 Create a PDF/A with all color and grayscale images converted to JPEG

```
ocrmypdf --output-type pdfa --pdfa-image-compression jpeg input.pdf output.pdf
```

6.1.5 Modify a file in place

The file will only be overwritten if OCRmyPDF is successful.

```
ocrmypdf myfile.pdf myfile.pdf
```

6.1.6 Correct page rotation

OCR will attempt to automatic correct the rotation of each page. This can help fix a scanning job that contains a mix of landscape and portrait pages.

```
ocrmypdf --rotate-pages myfile.pdf myfile.pdf
```

You can increase (decrease) the parameter `--rotate-pages-threshold` to make page rotation more (less) aggressive.

If the page is "just a little off horizontal", like a crooked picture, then you want `--deskew`. `--rotate-pages` is for when the cardinal angle is wrong.

6.1.7 OCR languages other than English

By default OCRmyPDF assumes the document is English.

```
ocrmypdf -l fra LeParisien.pdf LeParisien.pdf
ocrmypdf -l eng+fra Bilingual-English-French.pdf Bilingual-English-French.pdf
```

Language packs must be installed for all languages specified. See *Installing additional language packs*.

6.1.8 Produce PDF and text file containing OCR text

This produces a file named "output.pdf" and a companion text file named "output.txt".

```
ocrmypdf --sidecar output.txt input.pdf output.pdf
```

6.2 OCR images, not PDFs

If you are starting with images, you can just use Tesseract directly to convert images to PDFs:

```
tesseract my-image.jpg output-prefix pdf
```

```
# When there are multiple images
tesseract text-file-containing-list-of-image-filenames.txt output-prefix pdf
```

Tesseract's PDF output is quite good – OCRmyPDF uses it internally by default. However, OCRmyPDF has many features not available in Tesseract like like image processing, metadata control, and PDF/A generation.

Use a program like img2pdf to convert your images to PDFs, and then pipe the results to run ocrmypdf. The - tells ocrmypdf to read standard input.

```
img2pdf my-images*.jpg | ocrmypdf - myfile.pdf
```

`img2pdf` is recommended because it does an excellent job at generating PDFs without transcoding images.

For convenience, OCRmyPDF can also convert single images to PDFs on its own. If the resolution (dots per inch, DPI) of an image is not set or is incorrect, it can be overridden with `--image-dpi`. (As 1 inch is 2.54 cm, 1 dpi = 0.39 dpcm).

```
ocrmypdf --image-dpi 300 image.png myfile.pdf
```

If you have multiple images, you must use `img2pdf` to convert the images to PDF.

Note: ImageMagick `convert` can also convert a group of images to PDF, but in the author's experience it takes a long time, transcodes unnecessarily and gives poor results.

6.3 Image processing

OCRmyPDF perform some image processing on each page of a PDF, if desired. The same processing is applied to each page. It is suggested that the user review files after image processing as these commands might remove desirable content, especially from poor quality scans.

- `--rotate-pages` attempts to determine the correct orientation for each page and rotates the page if necessary.
- `--remove-background` attempts to detect and remove a noisy background from grayscale or color images. Monochrome images are ignored. This should not be used on documents that contain color photos as it may remove them.
- `--deskew` will correct pages were scanned at a skewed angle by rotating them back into place. Skew determination and correction is performed using Postl's variance of line sums algorithm as implemented in Leptonica.
- `--clean` uses unpaper to clean up pages before OCR, but does not alter the final output. This makes it less likely that OCR will try to find text in background noise.
- `--clean-final` uses unpaper to clean up pages before OCR and inserts the page into the final output. You will want to review each page to ensure that unpaper did not remove something important.
- `--mask-barcodes` will "cover up" any barcodes detected in the image of a page. Barcodes are known to confuse Tesseract OCR and interfere with the recognition of text on the same baseline as a barcode. The output file will contain the unaltered image of the barcode.

Note: In many cases image processing will rasterize PDF pages as images, potentially losing quality.

Warning: `--clean-final` and `-remove-background` may leave undesirable visual artifacts in some images where their algorithms have shortcomings. Files should be visually reviewed after using these options.

6.3.1 OCR and correct document skew (crooked scan)

Deskew:

```
ocrmypdf --deskew input.pdf output.pdf
```

Image processing commands can be combined. The order in which options are given does not matter. OCRmyPDF always applies the steps of the image processing pipeline in the same order (rotate, remove background, deskew, clean).

```
ocrmypdf --deskew --clean --rotate-pages input.pdf output.pdf
```

6.3.2 Don't actually OCR my PDF

If you set `--tesseract-timeout 0` OCRmyPDF will apply its image processing without performing OCR, if all you want to is to apply image processing or PDF/A conversion.

```
ocrmypdf --tesseract-timeout=0 --remove-background input.pdf output.pdf
```

6.3.3 Redo existing OCR

To redo OCR on a file OCRed with other OCR software or a previous version of OCRmyPDF and/or Tesseract, you may use the `--redo-ocr` argument. (Normally, OCRmyPDF will exit with an error if asked to modify a file with OCR.)

This may be helpful for users who want to take advantage of accuracy improvements in Tesseract 4.0 for files they previously OCRed with an earlier version of Tesseract and OCRmyPDF.

```
ocrmypdf --redo-ocr input.pdf output.pdf
```

This method will replace OCR without rasterizing, reducing quality or removing vector content. If a file contains a mix of pure digital text and OCR, digital text will be ignored and OCR will be replaced. As such this mode is incompatible with image processing options, since they alter the appearance of the file.

In some cases, existing OCR cannot be detected or replaced. Files produced by OCRmyPDF v2.2 or earlier, for example, are internally represented as having visible text with an opaque image drawn on top. This situation cannot be detected.

If `--redo-ocr` does not work, you can use `--force-ocr`, which will force rasterization of all pages, potentially reducing quality or losing vector content.

6.4 Improving OCR quality

The *Image processing* features can improve OCR quality.

Rotating pages and deskewing helps to ensure that the page orientation is correct before OCR begins. Removing the background and/or cleaning the page can also improve results. The `--oversample DPI` argument can be specified to resample images to higher resolution before attempting OCR; this can improve results as well.

OCR quality will suffer if the resolution of input images is not correct (since the range of pixel sizes that will be checked for possible fonts will also be incorrect).

6.5 PDF optimization

By default OCRmyPDF will attempt to perform lossless optimizations on the images inside PDFs after OCR is complete. Optimization is performed even if no OCR text is found.

The `--optimize N` (short form `-O`) argument controls optimization, where `N` ranges from 0 to 3. `--optimize 0` disables optimizations. `1` enables lossless optimizations that can be performed safely with no quality loss. `2` enables lossy optimizations such as image color quantizations. `3` enables more aggressive optimizations and targets a lower JPEG quality.

Optimization is improved when a JBIG2 encoder is available and when `pngquant` is installed. If either of these components are missing, then some types of images will not be optimized.

Currently optimization attempts to find more efficient encodings for images. The types of optimization available may expand over time. By default, OCRmyPDF compresses data streams inside PDFs, and will change inefficient encodings to more modern versions. A program like `qpdf` can be used to change encodings, e.g. to inspect the internals fo a PDF.

```
ocrmypdf --optimize 3 in.pdf out.pdf  # Make it small
```

Some users may consider enabling lossy JBIG2. See: *Lossy mode JBIG2*.

CHAPTER 7

Advanced features

7.1 Control of OCR options

OCRmyPDF provides many features to control the behavior of the OCR engine, Tesseract.

7.1.1 When OCR is skipped

If a page in a PDF seems to have text, by default OCRmyPDF will exit without modifying the PDF. This is to ensure that PDFs that were previously OCRed or were "born digital" rather than scanned are not processed.

If `--skip-text` is issued, then no OCR will be performed on pages that already have text. The page will be copied to the output. This may be useful for documents that contain both "born digital" and scanned content, or to use OCRmyPDF to normalize and convert to PDF/A regardless of their contents.

If `--redo-ocr` is issued, then a detailed text analysis is performed. Text is categorized as either visible or invisible. Invisible text (OCR) is stripped out. Then an image of each page is created with visible text masked out. The page image is sent for OCR, and any additional text is inserted as OCR. If a file contains a mix of text and bitmap images that contain text, OCRmyPDF will locate the additional text in images without disrupting the existing text.

If `--force-ocr` is issued, then all pages will be rasterized to images, discarding any hidden OCR text, and rasterizing any printable text. This is useful for redoing OCR, for fixing OCR text with a damaged character map (text is selectable but not searchable), and destroying redacted information. Any forms and vector graphics will be rasterized as well.

7.1.2 Time and image size limits

By default, OCRmyPDF permits tesseract to run for three minutes (180 seconds) per page. This is usually more than enough time to find all text on a reasonably sized page with modern hardware.

If a page is skipped, it will be inserted without OCR. If preprocessing was requested, the preprocessed image layer will be inserted.

If you want to adjust the amount of time spent on OCR, change `--tesseract-timeout`. You can also automatically skip images that exceed a certain number of megapixels with `--skip-big`. (A 300 DPI, 8.5×11" page is 8.4 megapixels.)

```
# Allow 300 seconds for OCR; skip any page larger than 50 megapixels
ocrmypdf --tesseract-timeout 300 --skip-big 50 bigfile.pdf output.pdf
```

7.1.3 Overriding default tesseract

OCRmyPDF checks the system `PATH` for the `tesseract` binary.

Some relevant environment variables that influence Tesseract's behavior include:

`TESSDATA_PREFIX`
: Overrides the path to Tesseract's data files. This can allow simultaneous installation of the "best" and "fast" training data sets. OCRmyPDF does not manage this environment variable.

`OMP_THREAD_LIMIT`
: Controls the number of threads Tesseract will use. OCRmyPDF will manage this environment if it is not already set. (Currently, it will set it to 1 because this gives the best results in testing.)

For example, if you have a development build of Tesseract don't wish to use the system installation, you can launch OCRmyPDF as follows:

```
env \
    PATH=/home/user/src/tesseract/api:$PATH \
    TESSDATA_PREFIX=/home/user/src/tesseract \
    ocrmypdf input.pdf output.pdf
```

In this example `TESSDATA_PREFIX` is required to redirect Tesseract to an alternate folder for its "tessdata" files.

7.1.4 Overriding other support programs

In addition to tesseract, OCRmyPDF uses the following external binaries:

- `gs` (Ghostscript)
- `unpaper`
- `qpdf`

In each case OCRmyPDF will search the `PATH` environment variable to locate the binaries.

7.1.5 Changing tesseract configuration variables

You can override tesseract's default control parameters with a configuration file.

As an example, this configuration will disable Tesseract's dictionary for current language. Normally the dictionary is helpful for interpolating words that are unclear, but it may interfere with OCR if the document does not contain many words (for example, a list of part numbers).

Create a file named "no-dict.cfg" with these contents:

```
load_system_dawg 0
language_model_penalty_non_dict_word 0
language_model_penalty_non_freq_dict_word 0
```

then run ocrmypdf as follows (along with any other desired arguments):

```
ocrmypdf --tesseract-config no-dict.cfg input.pdf output.pdf
```

Warning: Some combinations of control parameters will break Tesseract or break assumptions that OCRmyPDF makes about Tesseract's output.

7.2 Changing the PDF renderer

rasterizing Converting a PDF to an image for display.

rendering Creating a new PDF from other data (such as an existing PDF).

OCRmyPDF has these PDF renderers: `sandwich` and `hocr`. The renderer may be selected using `--pdf-renderer`. The default is `auto` which lets OCRmyPDF select the renderer to use. Currently, `auto` always selects `sandwich`.

7.2.1 The `sandwich` renderer

The `sandwich` renderer uses Tesseract's new text-only PDF feature, which produces a PDF page that lays out the OCR in invisible text. This page is then "sandwiched" onto the original PDF page, allowing lossless application of OCR even to PDF pages that contain other vector objects.

Currently this is the best renderer for most uses, however it is implemented in Tesseract so OCRmyPDF cannot influence it. Currently some problematic PDF viewers like Mozilla PDF.js and macOS Preview have problems with segmenting its text output, and mightrunseveralwordstogether.

When image preprocessing features like `--deskew` are used, the original PDF will be rendered as a full page and the OCR layer will be placed on top.

7.2.2 The `hocr` renderer

The `hocr` renderer works with older versions of Tesseract. The image layer is copied from the original PDF page if possible, avoiding potentially lossy transcoding or loss of other PDF information. If preprocessing is specified, then the image layer is a new PDF.

Unlike `sandwich` this renderer is implemented within OCRmyPDF; anyone looking to customize how OCR is presented should look here. A major disadvantage of this renderer is it not capable of correctly handling text outside the Latin alphabet. Pull requests to improve the situation are welcome.

Currently, this renderer has the best compatibility with Mozilla's PDF.js viewer.

This works in all versions of Tesseract.

7.2.3 The `tesseract` renderer

The `tesseract` renderer was removed. OCRmyPDF's new approach to text layer grafting makes it functionally equivalent to `sandwich`.

7.3 Return code policy

OCRmyPDF writes all messages to `stderr`. `stdout` is reserved for piping output files. `stdin` is reserved for piping input files.

The return codes generated by the OCRmyPDF are considered part of the stable user interface. They may be imported from `ocrmypdf.exceptions`.

Table 1: Return codes

Code	Name	Interpretation
0	`ExitCode.ok`	Everything worked as expected.
1	`ExitCode.bad_args`	Invalid arguments, exited with an error.
2	`ExitCode.input_file`	The input file does not seem to be a valid PDF.
3	`ExitCode.missing_dependency`	An external program required by OCRmyPDF is missing.
4	`ExitCode.invalid_output_pdf`	An output file was created, but it does not seem to be a valid PDF. The file will be available.
5	`ExitCode.file_access_error`	The user running OCRmyPDF does not have sufficient permissions to read the input file and write the output file.
6	`ExitCode.already_done_ocr`	The file already appears to contain text so it may not need OCR. See output message.
7	`ExitCode.child_process_error`	An error occurred in an external program (child process) and OCRmyPDF cannot continue.
8	`ExitCode.encrypted_pdf`	The input PDF is encrypted. OCRmyPDF does not read encrypted PDFs. Use another program such as `qpdf` to remove encryption.
9	`ExitCode.invalid_config`	A custom configuration file was forwarded to Tesseract using `--tesseract-config`, and Tesseract rejected this file.
10	`ExitCode.pdfa_conversion_failed`	A valid PDF was created, PDF/A conversion failed. The file will be available.
15	`ExitCode.other_error`	Some other error occurred.
130	`ExitCode.ctrl_c`	The program was interrupted by pressing Ctrl+C.

7.4 Debugging the intermediate files

OCRmyPDF normally saves its intermediate results to a temporary folder and deletes this folder when it exits, whether it succeeded or failed.

If the `-k` argument is issued on the command line, OCRmyPDF will keep the temporary folder and print the location, whether it succeeded or failed (provided the Python interpreter did not crash). An example message is:

The organization of this folder is an implementation detail and subject to change between releases. However the general organization is that working files on a per page basis have the page number as a prefix (starting with page 1), an infix indicates the processing stage, and a suffix indicates the file type. Some important files include:

- `.page.png` - what the input page looks like
- `.image` - the image we will show the user if we are in a mode that changes the final appearance; may be in one of several image formats
- `.text.pdf` - the OCR file; this will load as a blank page but should have visible text if checked with a tool like pdftotext or pdfminder.six
- `.ocr.png` - the file that is sent to Tesseract for OCR; depending on arguments this may differ from the presentation image

- `layers.rendered.pdf` - the composite PDF, before metadata repair and optimization
- `images/*` - images extracted during the optimization process; here the prefix indicates a PDF object ID not a page number

CHAPTER 8

Batch processing

This article provides information about running OCRmyPDF on multiple files or configuring it as a service triggered by file system events.

8.1 Batch jobs

Consider using the excellent GNU Parallel to apply OCRmyPDF to multiple files at once.

Both `parallel` and `ocrmypdf` will try to use all available processors. To maximize parallelism without overloading your system with processes, consider using `parallel -j 2` to limit parallel to running two jobs at once.

This command will run all ocrmypdf all files named `*.pdf` in the current directory and write them to the previous created `output/` folder. It will not search subdirectories.

The `--tag` argument tells parallel to print the filename as a prefix whenever a message is printed, so that one can trace any errors to the file that produced them.

```
parallel --tag -j 2 ocrmypdf '{}' 'output/{}' ::: *.pdf
```

OCRmyPDF automatically repairs PDFs before parsing and gathering information from them. If you are already repairing PDFs with `qpdf` prior to attempting OCR, or you can use `--skip-repair` to skip this step. It may improve performance for large files, since repairing PDFs is single-threaded.

8.2 Directory trees

This will walk through a directory tree and run OCR on all files in place, printing the output in a way that makes

```
find . --printf '%p' -name '*.pdf' -exec ocrmypdf '{}' '{}' \;
```

This only runs one `ocrmypdf` process at a time. This variation uses `find` to create a directory list and `parallel` to parallelize runs of `ocrmypdf`, again updating files in place.

```
find . -name '*.pdf' | parallel --tag -j 2 ocrmypdf '{}' '{}'
```

8.2.1 Sample script

This user contributed script also provides an example of batch processing.

```
#!/usr/bin/env python3
# Walk through directory tree, replacing all files with OCR'd version
# Contributed by DeliciousPickle@github

import logging
import os
import subprocess
import sys

script_dir = os.path.dirname(os.path.realpath(__file__))
print(script_dir + '/ocr-tree.py: Start')

if len(sys.argv) > 1:
    start_dir = sys.argv[1]
else:
    start_dir = '.'

if len(sys.argv) > 2:
    log_file = sys.argv[2]
else:
    log_file = script_dir + '/ocr-tree.log'

logging.basicConfig(
                level=logging.INFO, format='%(asctime)s %(message)s',
                filename=log_file, filemode='w')

for dir_name, subdirs, file_list in os.walk(start_dir):
    logging.info('\n')
    logging.info(dir_name + '\n')
    os.chdir(dir_name)
    for filename in file_list:
        file_ext = os.path.splitext(filename)[1]
        if file_ext == '.pdf':
            full_path = dir_name + '/' + filename
            print(full_path)
            cmd = ["ocrmypdf",  "--deskew", filename, filename]
            logging.info(cmd)
            proc = subprocess.Popen(
                cmd, stdout=subprocess.PIPE, stderr=subprocess.STDOUT)
            result = proc.stdout.read()
            if proc.returncode == 6:
                print("Skipped document because it already contained text")
            elif proc.returncode == 0:
                print("OCR complete")
            logging.info(result)
```

8.2.2 API

OCRmyPDF is currently supported as a command line interface. This means that even if you are using OCRmyPDF in a Python script, you should run it in a subprocess rather importing the ocrmypdf package.

The reason for this limitation is that the ruffus library that OCRmyPDF depends on is unfortunately not reentrant. OCRmyPDF works by defining each operation it does as a ruffus task that takes one or more files as input and generates one or more files as output. As such ruffus is fairly fundamental.

(If you find individual functions implemented in OCRmyPDF useful (such as `ocrmypdf.pdfinfo`), you can use these if you wish to.)

8.2.3 Synology DiskStations

Synology DiskStations (Network Attached Storage devices) can run the Docker image of OCRmyPDF if the Synology Docker package is installed. Attached is a script to address particular quirks of using OCRmyPDF on one of these devices.

This is only possible for x86-based Synology products. Some Synology products use ARM or Power processors and do not support Docker. Further adjustments might be needed to deal with the Synology's relatively limited CPU and RAM.

```
#!/bin/env python3
# Contributed by github.com/Enantiomerie

# script needs 2 arguments
# 1. source dir with *.pdf - default is location of script
# 2. move dir where *.pdf and *_OCR.pdf are moved to

import logging
import os
import subprocess
import sys
import time
import shutil

script_dir = os.path.dirname(os.path.realpath(__file__))
timestamp = time.strftime("%Y-%m-%d-%H%M_")
log_file = script_dir + '/' + timestamp + 'ocrmypdf.log'
logging.basicConfig(level=logging.INFO, format='%(asctime)s %(message)s',␣
↪filename=log_file, filemode='w')

if len(sys.argv) > 1:
    start_dir = sys.argv[1]
else:
    start_dir = '.'

for dir_name, subdirs, file_list in os.walk(start_dir):
    logging.info('\n')
    logging.info(dir_name + '\n')
    os.chdir(dir_name)
    for filename in file_list:
        file_ext = os.path.splitext(filename)[1]
        if file_ext == '.pdf':
            full_path = dir_name + '/' + filename
            file_noext = os.path.splitext(filename)[0]
            timestamp_OCR = time.strftime("%Y-%m-%d-%H%M_OCR_")
```

(continues on next page)

(continued from previous page)

```
            filename_OCR = timestamp_OCR + file_noext + '.pdf'
            docker_mount = dir_name + ':/home/docker'
# create string for pdf processing
# diskstation needs a user:group docker:docker. find uid:gid of your diskstation␣
↪docker:docker with id docker.
# use this uid:gid in -u flag
# rw rights for docker:docker at source dir are also necessary
# the script is processed as root user via chron
            cmd = ['docker', 'run', '--rm', '-v', docker_mount, '-u=1030:65538',
↪'jbarlow83/ocrmypdf', , '--deskew' , filename, filename_OCR]
            logging.info(cmd)
            proc = subprocess.Popen(cmd, stdout=subprocess.PIPE, stderr=subprocess.
↪STDOUT)
            result = proc.stdout.read()
            logging.info(result)
            full_path_OCR = dir_name + '/' + filename_OCR
            os.chmod(full_path_OCR, 0o666)
            os.chmod(full_path, 0o666)
            full_path_OCR_archive = sys.argv[2]
            full_path_archive = sys.argv[2] + '/no_ocr'
            shutil.move(full_path_OCR,full_path_OCR_archive)
            shutil.move(full_path, full_path_archive)
logging.info('Finished.\n')
```

8.2.4 Huge batch jobs

If you have thousands of files to work with, contact the author. Consulting work related to OCRmyPDF helps fund this open source project and all inquiries are appreciated.

8.3 Hot (watched) folders

To set up a "hot folder" that will trigger OCR for every file inserted, use a program like Python watchdog (supports all major OS).

One could then configure a scanner to automatically place scanned files in a hot folder, so that they will be queued for OCR and copied to the destination.

```
pip install watchdog
```

watchdog installs the command line program `watchmedo`, which can be told to run `ocrmypdf` on any .pdf added to the current directory (`.`) and place the result in the previously created `out/` folder.

```
cd hot-folder
mkdir out
watchmedo shell-command \
        --patterns="*.pdf" \
        --ignore-directories \
        --command='ocrmypdf "${watch_src_path}" "out/${watch_src_path}" ' \
        .  # don't forget the final dot
```

For more complex behavior you can write a Python script around to use the watchdog API.

On file servers, you could configure watchmedo as a system service so it will run all the time.

8.3.1 Caveats

- `watchmedo` may not work properly on a networked file system, depending on the capabilities of the file system client and server.
- This simple recipe does not filter for the type of file system event, so file copies, deletes and moves, and directory operations, will all be sent to ocrmypdf, producing errors in several cases. Disable your watched folder if you are doing anything other than copying files to it.
- If the source and destination directory are the same, watchmedo may create an infinite loop.
- On BSD, FreeBSD and older versions of macOS, you may need to increase the number of file descriptors to monitor more files, using `ulimit -n 1024` to watch a folder of up to 1024 files.

8.3.2 Alternatives

- Watchman is a more powerful alternative to `watchmedo`.

CHAPTER 9

PDF security issues

OCRmyPDF should only be used on PDFs you trust. It is not designed to protect you against malware.

Recognizing that many users have an interest in handling PDFs and applying OCR to PDFs they did not generate themselves, this article discusses the security implications of PDFs and how users can protect themselves.

The disclaimer applies: this software has no warranties of any kind.

9.1 PDFs may contain malware

PDF is a rich, complex file format. The official PDF 1.7 specification, ISO 32000:2008, is hundreds of pages long and references several annexes each of which are similar in length. PDFs can contain video, audio, XML, JavaScript and other programming, and forms. In some cases, they can open internet connections to pre-selected URLs. All of these possible attack vectors.

In short, PDFs may contain viruses.

This article describes a high-paranoia method which allows potentially hostile PDFs to be viewed and rasterized safely in a disposable virtual machine. A trusted PDF created in this manner is converted to images and loses all information making it searchable and losing all compression. OCRmyPDF could be used restore searchability.

9.2 How OCRmyPDF processes PDFs

OCRmyPDF must open and interpret your PDF in order to insert an OCR layer. First, it runs all PDFs through pikepdf, a library based on qpdf, a program that repairs PDFs with syntax errors. This is done because, in the author's experience, a significant number of PDFs in the wild especially those created by scanners are not well-formed files. qpdf makes it more likely that OCRmyPDF will succeed, but offers no security guarantees. qpdf is also used to split the PDF into single page PDFs.

Finally, OCRmyPDF rasterizes each page of the PDF using Ghostscript in `-dSAFER` mode.

Depending on the options specified, OCRmyPDF may graft the OCR layer into the existing PDF or it may essentially reconstruct ("re-fry") a visually identical PDF that may be quite different at the binary level. That said, OCRmyPDF is not a tool designed for sanitizing PDFs.

9.3 Using OCRmyPDF online or as a service

OCRmyPDF should not be deployed as a public-facing service, such as a website where a potential attacker could upload a PDF of their choice for OCR. OCRmyPDF is not designed to be secure against PDF malware. Another concern is PDFs specifically designed to be a denial of service attack: PDFs can contain recursive data structures that sometimes send parsers into infinite loops, and issue complex graphics drawing commands.

Setting aside these concerns, a side effect of OCRmyPDF is it may incidentally sanitize PDFs that contain malware. It runs `qpdf` to repair the PDF, which could correct malformed PDF structures that are part of an attack. When PDF/A output is selected (the default), the input PDF is partially reconstructed by Ghostscript. When `--force-ocr` is used, all pages are rasterized and reconverted to PDF, which could remove malware in embedded images. No guarantees.

OCRmyPDF should be relatively safe to use in a trusted intranet, with some considerations:

9.3.1 Limiting CPU usage

OCRmyPDF will attempt to use all available CPUs and storage, so executing `nice ocrmypdf` or limiting the number of jobs with the `-j` argument may ensure the server remains available. Another option would be run OCRmyPDF jobs inside a Docker container, a virtual machine, or a cloud instance, which can impose its own limits on CPU usage and be terminated "from orbit" if it fails to complete.

9.3.2 Temporary storage requirements

OCRmyPDF will use a large amount of temporary storage for its work, proportional to the total number of pixels needed to rasterize the PDF. The raster image of a 8.5×11" color page at 300 DPI takes 25 MB uncompressed; OCRmyPDF saves its intermediates as PNG, but that still means it requires about 9 MB per intermediate based on average compression ratios. Multiple intermediates per page are also required, depending on the command line given. A rule of thumb would be to allow 100 MB of temporary storage per page in a file – meaning that a small cloud servers or small VM partitions should be provisioned with plenty of extra space, if say, a 500 page file might be sent.

To check temporary storage usage on actual files, run `ocrmypdf -k ...` which will preserve and print the path to temporary storage when the job is done.

To change where temporary files are stored, change the `TMPDIR` environment variable for ocrmypdf's environment. (Python's `tempfile.gettempdir()` returns the root directory in which temporary files will be stored.) For example, one could redirect `TMPDIR` to a large RAM disk to avoid wear on HDD/SSD and potentially improve performance. On Amazon Web Services, `TMPDIR` can be set to empheral storage.

9.3.3 Timeouts

To prevent excessively long OCR jobs consider setting `--tesseract-timeout` and/or `--skip-big` arguments. `--skip-big` is particularly helpful if your PDFs include documents such as reports on standard page sizes with large images attached - often large images are not worth OCR'ing anyway.

9.3.4 Commercial alternatives

The author also provides professional services that include OCR and building databases around PDFs, and is happy to provide consultation.

Abbyy Cloud OCR is a viable commercial alternative with a web services API.

9.4 Password protection, digital signatures and certification

Password protected PDFs usually have two passwords, and owner and user password. When the user password is set to empty, PDF readers will open the file automatically and marked it as "(SECURED)". While not as reliable as a digital signature, this indicates that whoever set the password approved of the file at that time. When the user password is set, the document cannot be viewed without the password.

Either way, OCRmyPDF does not remove passwords from PDFs and exits with an error on encountering them.

`qpdf`, one of OCRmyPDF's dependencies, can remove passwords. If the owner and user password are set, a password is required for `qpdf`. If only the owner password is set, then the password can be stripped, even if one does not have the owner password.

After OCR is applied, password protection is not permitted on PDF/A documents but the file can be converted to regular PDF.

Many programs exist which are capable of inserting an image of someone's signature. On its own, this offers no security guarantees. It is trivial to remove the signature image and apply it to other files. This practice offers no real security.

Important documents can be digitally signed and certified to attest to their authorship. OCRmyPDF cannot do this. Open source tools such as pdfbox (Java) have this capability as does Adobe Acrobat.

CHAPTER 10

Common error messages

10.1 Page already has text

```
ERROR -    1: page already has text! - aborting (use --force-ocr to force OCR)
```

You ran ocrmypdf on a file that already contains printable text or a hidden OCR text layer (it can't quite tell the difference). You probably don't want to do this, because the file is already searchable.

As the error message suggests, your options are:

- `ocrmypdf --force-ocr` to *rasterize* all vector content and run OCR on the images. This is useful if a previous OCR program failed, or if the document contains a text watermark.
- `ocrmypdf --skip-text` to skip OCR and other processing on any pages that contain text. Text pages will be copied into the output PDF without modification.

10.2 Input file 'filename' is not a valid PDF

OCRmyPDF passes files through qpdf, a program that fixes errors in PDFs, before it tries to work on them. In most cases this happens because the PDF is corrupt and truncated (incomplete file copying) and not much can be done.

You can try rewriting the file with Ghostscript or pdftk:

- `gs -o output.pdf -dSAFER -sDEVICE=pdfwrite input.pdf`
- `pdftk input.pdf cat output output.pdf`

Sometimes Acrobat can repair PDFs with its Preflight tool.

CHAPTER 11

Indices and tables

- genindex
- modindex
- search

Index

E