CodePDF

Creates PDF/HTML files from code/markdown files.

Dependencies:

These are system package dependencies.

- **Python 3+** (`python3`): This program uses python 3 features, and is not compatible with Python 2.
- **WKHtmlToPDF** (`wkhtmltopdf`): Converts HTML to PDF, and is required by `pdfkit`.

Python package dependencies:

These packages can be installed with `pip`.

- **Docopt** (`docopt`): Used for command-line argument parsing.
- **Markdown** (`markdown`): Used for converting markdown files.
- **PdfKit** (`pdfkit`): Used for converting html to pdf.
- **Pygments** (`pygments`): Used for converting code files.

Installation:

I recommend symlinking this script somewhere in your `$PATH`:

```
git clone https://github.com/welbornprod/codepdf.git
cd codepdf
ln -s "$PWD/codepdf.py" ~/.local/bin/codepdf
```

Command line help:

```
Usage:
    codepdf -h | -S | -v
    codepdf [FILE...] [-f] [-H] [-l] [-n] [-o file]
            [-s style] [-t title] [-D]

Options:
    FILE                     : File names to convert, or - for stdin.
                               If no names are given, stdin is used.
    -D,--debug               : Print some debug info while running.
    -f,--forcemd             : Highlight markdown syntax, instead of
                               converting to HTML.
    -h,--help                : Show this help message.
    -H,--html                : Output in HTML instead of PDF.
                               Using .htm or .html as the output file
                               extension will automatically set this flag.
    -l,--linenumbers         : Use line numbers.
    -n,--noconfig            : Ignore config file settings.
    -o file,--out file       : Output file name.
                               Default: <input_basename>.pdf
    -s name,--style name     : Pygments style name to use for code files.
                               Default: default
    -S,--styles              : Print all known pygments styles.
    -t title,--title title   : Title for the PDF.
                               Default: <input_filename>
    -v,--version             : Show version.
```

Config:

The `forcemd`, `html`, `linenumbers`, `style`, and `title` options can be set permanently in a config file (`codepdf.json`).

The file can be located in the current working dir (`$PWD`), your home dir (`~`), or next to `codepdf.py`.

The directories are tried in that order.

Single line javascript comments are acceptable.

Example config (codepdf.json):

```
{
    // Syntax highlight markdown files instead of converting to HTML.
    "forcemd": false,
    // Use HTML output instead of PDF.
    "html": false,
    // Use line numbers for source files and readme code sections.
    "linenumbers": false,
    // Pygments style to use.
    "style": "default",
    // Title for the output PDF/HTML.
    "title": null
}
```

Examples:

example.html is an HTML file that was created by running:

```
codepdf README.md requirements.txt codepdf.py -l -s monokai -o example.html
```

This is the same HTML that is used to create the PDF file.

example.pdf is a PDF file that was created by running:

```
codepdf README.md requirements.txt codepdf.py -l -s monokai -o example.pdf
```

---

requirements.txt

```
Colr>=0.2.5
docopt>=0.6.2
Markdown>=2.6.6
pdfkit>=0.5.0
Pygments>=2.1.3
```

---

codepdf.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

""" codepdf.py
    Convert code/text files to pdf.
    -Christopher Welborn 06-13-2016
"""
# print_function just to say "don't use python 2."
from __future__ import print_function
import inspect
import json
import os
import sys
try:
    from contextlib import suppress
except ImportError as ex:
    print('Error importing contextlib.suppress: {}'.format(ex))
    if sys.version_info.major < 3:
        # Better message than 'cannot import name suppress'
        print(
            '\n'.join((
                '\nCodePDF only works with Python 3+.',
                '\nCurrent python version:\n  {}'.format(
                    sys.version.replace('\n', '\n  ')
                )
            )),
            file=sys.stderr
        )

    sys.exit(1)

# Third-party libs.
try:
    from colr import (
        auto_disable as colr_auto_disable,
        Colr as C
    )
    from docopt import docopt
    from markdown import markdown
    from markdown.extensions.codehilite import CodeHiliteExtension
    from markdown.extensions.fenced_code import FencedCodeExtension
    from markdown.extensions.sane_lists import SaneListExtension
    from pdfkit import from_string as pdf_from_string
    from pygments import highlight, lexers, formatters, styles
    from pygments.util import ClassNotFound
except ImportError as eximpcolr:
    print(
```

```python
        '\n'.join((
            'Failed to import {pname}, you may need to install it:',
            '    pip install {exc.name}',
            'Original error:',
            '    {exc.msg}'
        )).format(
            pname=exc.name.title(),
            exc=exc
        ),
        file=sys.stderr
    )
    sys.exit(1)

# Disable colors when piping output.
colr_auto_disable()

NAME = 'CodePDF'
VERSION = '0.0.5'
VERSIONSTR = '{} v. {}'.format(NAME, VERSION)
SCRIPT = os.path.split(os.path.abspath(sys.argv[0]))[1]
SCRIPTDIR = os.path.abspath(sys.path[0])

# Global debug flag, set with --debug.
DEBUG = False
# File name to trigger reading from stdin.
STDIN_NAME = '-'
# Default pygments style.
DEFAULT_STYLE = 'default'
# Default pygments lexer, when it can't be detected.
DEFAULT_LEXER = 'text'
# Class name for each file's div.
DIV_CLASS = 'hilight'

USAGESTR = """{versionstr}
    Usage:
        {script} -h | -S | -v
        {script} [FILE...] [-f] [-H] [-l] [-n] [-o file]
                 [-s style] [-t title] [-D]

    Options:
        FILE                      : File names to convert, or {stdin} for stdin.
                                    If no names are given, stdin is used.
        -D,--debug                : Print some debug info while running.
        -f,--forcemd              : Highlight markdown syntax, instead of
                                    converting to HTML.
        -h,--help                 : Show this help message.
        -H,--html                 : Output in HTML instead of PDF.
                                    Using .htm or .html as the output file
                                    extension will automatically set this flag.
        -l,--linenumbers          : Use line numbers.
        -n,--noconfig             : Ignore config file settings.
        -o file,--out file        : Output file name.
                                    Default: <input_basename>.pdf
        -s name,--style name      : Pygments style name to use for code files.
                                    Default: {default_style}
        -S,--styles               : Print all known pygments styles.
        -t title,--title title    : Title for the PDF.
                                    Default: <input_filename>
        -v,--version              : Show version.
""".format(
    default_style=DEFAULT_STYLE,
    script=SCRIPT,
    stdin=STDIN_NAME,
    versionstr=VERSIONSTR
)


def main(argd):
    """ Main entry point, expects doctopt arg dict as argd. """
    global DEBUG
    DEBUG = argd['--debug']
    # Load user config, if applicable.
    argd = load_config(argd)

    if argd['--styles']:
        return print_styles()

    filenames = argd['FILE'] or [STDIN_NAME]
    html_mode = argd['--html']
    outname = get_output_name(
        filenames,
        output_name=argd['--out'],
        html_mode=html_mode,
    )
    # Check for user-provided .html output file.
    if not html_mode:
        html_mode = outname.lower().endswith(('.htm', '.html'))

    success = convert_files(
        argd['FILE'] or [STDIN_NAME],
        argd['--out'] or get_output_name(filenames),
        stylename=argd['--style'],
        linenos=argd['--linenumbers'],
        title=argd['--title'],
        force_highlight=argd['--forcemd'],
        html_mode=html_mode,
    )
    if success:
        print(outname)
        return 0
    return 1


def build_html(body, styles=None, title=None):
    """ Try to build a somewhat-sane html page from a body and style-defs. """
    if not styles:
        styles = ['body {font-family: sans-serif;}']
    else:
        styles = list(styles)
        styles.insert(0, 'body {font-family: sans-serif;}')

    styles.append('\n'.join((
        'hr {',
        'border-style: hidden;',
        'height: 2px;',
        'background: #f1f1f1;',
        'margin-top: 25px;',
        '}',
    )))
    return '\n'.join((
        '<html>',
        '<head>',
        '<title>{}</title>'.format(title or ''),
        '<style type="text/css">',
        '\n'.join(styles),
        '</style>',
        '</head>',
        '<body>',
        body,
        '</body>',
        '</html>'
    ))


def convert_files(
        filenames, outputname,
        stylename=None, linenos=False,
        title=None, force_highlight=False, html_mode=False):
    """ Convert all files into a single PDF. """
    stylename = stylename or DEFAULT_STYLE
    debug(
        '\n'.join((
            'Converting files:\n    {}'.format(
                '\n    '.join(os.path.split(s)[-1] for s in filenames)
            ),
            'Output file: {outfile}',
            '     Forced: {forced}',
            '    LineNos: {linenos}',
            '      Style: {style}',
            '      Title: {title}',
        )).format(
            outfile=outputname,
            forced=force_highlight,
            linenos=linenos,
            style=stylename,
            title=title,
        )
    )
    htmlcontent = []
    styledefs = []
    for i, filename in enumerate(filenames):
        titletext = title or os.path.split(filename)[-1]
        if titletext in (STDIN_NAME,):
            titletext = 'stdin'
        formatter = get_formatter(
            stylename=stylename,
            linenos=linenos,
            title=titletext,
        )
        if not styledefs:
            styledefs.append(formatter.get_style_defs())
        htmlcontent.append(
            convert_to_html_div(
                filename,
                formatter,
                stylename=stylename,
                linenos=linenos,
                force_highlight=force_highlight
            )
        )
    allcontent = build_html(
        '<hr class="nv">'.join(htmlcontent),
            styles=styledefs,
            title=titletext
        )
        if html_mode:
            debug('Writing HTML to file...')
            with open(outputname, 'w') as f:
                f.write(allcontent)
        return True

        debug('Converting to PDF...')
        return pdf_from_string(
        allcontent,
        outputname,
        options={'--title': titletext, '--quiet': ''}
        )


def convert_hilight(filename, formatter):
        """ Highlight a file with pygments, and return the resulting HTML div. """
        displayname, content = get_file_content(filename)
        lexer = get_file_lexer(filename, content)
        debug('Highlighting: {}'.format(displayname))
        linkid = get_elem_id(displayname)
        return '\n'.join((
        '<div class="file">',
        get_permalink_html(linkid),
        '<h2 id="{}" style="display: inline-block">{}</h2>'.format(
                linkid,
                displayname
        ),
        '<div class="{}">'.format(DIV_CLASS),
        highlight(content, lexer, formatter),
        '</div>',
        '</div>'
        ))


def convert_markdown(filename, stylename=None, linenos=False):
        """ Convert a markdown file to an HTML div, and return the result. """
        displayname, content = get_file_content(filename)
        stylename = stylename.lower() if stylename else DEFAULT_STYLE
        debug('Converting MD: {}'.format(displayname))
        hilighter = CodeHiliteExtension(
        pygments_style=stylename,
        linenums=linenos,
        noclasses=True,
        css_class='hilight',
        )
        return '\n'.join((
        '<div class="markdown">',
        markdown(
                content,
                output_format='html5',
                extensions=[
                    hilighter,
                    FencedCodeExtension(),
                    SaneListExtension(),
                ]
        ),
        '</div>'
        ))


def convert_to_html_div(
        filename, formatter,
        stylename=None, linenos=False, force_highlight=False):
        """ Convert a file to an html div.
        The conversion method depends on the file extension.
        build_html() should be used with the content returned here.
        """
        if (not force_highlight) and filename.endswith(('.md', '.markdown')):
        return convert_markdown(
                filename,
                stylename=stylename,
                linenos=linenos
        )
        return convert_hilight(filename, formatter)


def debug(*args, **kwargs):
        """ Print a message only if DEBUG is truthy. """
        if not (DEBUG and args):
        return None

        # Include parent class name when given.
        parent = kwargs.get('parent', None)
        with suppress(KeyError):
        kwargs.pop('parent')

        # Go back more than once when given.
        backlevel = kwargs.get('back', 1)
        with suppress(KeyError):
        kwargs.pop('back')

        frame = inspect.currentframe()
        # Go back a number of frames (usually 1).
        while backlevel > 0:
        frame = frame.f_back
        backlevel -= 1
        fname = os.path.split(frame.f_code.co_filename)[-1]
        lineno = frame.f_lineno
        if parent:
        func = '{}.{}'.format(parent.__class__.__name__, frame.f_code.co_name)
        else:
        func = frame.f_code.co_name

        lineinfo = '{}:{} {}: '.format(
        C(fname, 'yellow'),
        C(str(lineno).ljust(4), 'blue'),
        C().join(C(func, 'magenta'), '()').ljust(20)
        )
        # Patch args to stay compatible with print().
        pargs = list(C(a, 'green').str() for a in args)
        pargs[0] = ''.join((lineinfo, pargs[0]))
        print(*pargs, **kwargs)


def get_elem_id(s):
        """ Transform a file name or text into a slug, usable for an element id.
        Removes non alpha-numeric characters, replaces spaces with -.
        """
        return '-'.join(
        ''.join(c for c in word if c.isalnum())
        for word in s.split()
        ).lower()


def get_file_content(filename):
        """ Returns a tuple of (display_name, content), handling stdin if
        STDIN_NAME is used.
        """
        if filename in (STDIN_NAME,):
        return 'stdin', read_stdin()

        with open(filename, 'r') as f:
        content = f.read()
        return os.path.split(filename)[-1], content


def get_file_lexer(filename, content):
        """ Try to get a lexer by file extension, guess by content if that fails.
        """
        try:
        # Pygments sometimes returns a weird lexer for .txt files.
        if filename.lower().endswith('.txt'):
                lexer = lexers.get_lexer_by_name('text')
                debug('Lexer forced by extension: {:>20} -> {}'.format(
                    lexer.name,
                    filename,
                ))
        else:
                lexer = lexers.get_lexer_for_filename(filename)
                debug('Lexer chosen by file name: {:>20} -> {}'.format(
                    lexer.name,
                    filename,
                ))
        except ClassNotFound:
        try:
                # Guess by content.
                lexer = lexers.guess_lexer(content)
                debug('Lexer guessed by content:  {:>20} -> {}'.format(
                    lexer.name,
                    filename,
                ))
        except ClassNotFound:
                # Fall back to default lexer.
                lexer = lexers.get_lexer_by_name(DEFAULT_LEXER)
                debug('Lexer set to default:      {:>20} -> {}'.format(
                    lexer.name,
                    filename,
                ))
        return lexer


def get_formatter(stylename=None, linenos=False, title=None, full=False):
        """ Get an HTMLFormatter from pygments. """
        stylename = stylename.lower() if stylename else DEFAULT_STYLE
        try:
        formatter = formatters.HtmlFormatter(
                cssclass=DIV_CLASS,
                linenos='inline' if linenos is True else linenos,
                style=stylename,
                full=full,
                title=title
            )
        except ClassNotFound:
            raise InvalidArg(
                '\n'.join((
                    'Unknown style name: {style}',
                    'Run `{script} --styles` to see a list of style names.',
                )).format(
                    style=stylename,
                    script=SCRIPT,
                )
            )
        return formatter


    def get_permalink_html(linkid):
        """ Return HTML needed to build a permalink link/icon for a header. """
        svg = """
        <svg style="vertical-align: middle; display: inline;"
        height="16" version="1.1" viewBox="0 0 16 16" width="16">
        <path d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0
        1.41-.91 2.72-2 3.25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98
        0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 2.5S13.98 12 13 12H9c-.98
        0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31
        7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z">
        </path></svg>
        """
        return '\n'.join((
            '<a href="#{}" style="text-decoration: none;">'.format(linkid),
            svg,
            '</a>'
        ))


    def get_output_name(filenames, output_name=None, html_mode=False):
        """ Determine output file name to use when the user hasn't given one. """
        if output_name:
            # Short-circuit auto-name-detection.
            return output_name

        inputname = filenames[0]
        if inputname == '-':
            inputname = 'stdin'
        parentdir, basename = os.path.split(inputname)
        if not parentdir:
            parentdir = os.getcwd()
        return '{name}{ext}'.format(
            name=os.path.join(parentdir, os.path.splitext(basename)[0]),
            ext='.html' if html_mode else '.pdf'
        )


    def load_config(argd):
        """ Load config from ~/.codepdf.json or ./.codepdf.json.
            Override argd values with user config.
        """
        if argd['--noconfig']:
            debug('Config ignored.')
            return argd

        filename = 'codepdf.json'
        for trydir in (os.getcwd(), os.path.expanduser('~'), SCRIPTDIR):
            filepath = os.path.join(trydir, filename)
            if os.path.exists(filepath):
                debug('Found config file: {}'.format(filepath))
                break
        else:
            debug('No config file found: {}'.format(filename))
            return argd

        with open(filepath, 'r') as f:
            content = ''.join(l for l in f if not l.strip().startswith('//'))

        try:
            rawconfig = json.loads(content)
        except ValueError as ex:
            raise ConfigError('Unable to load config from: {}\n{}'.format(
                filepath,
                ex
            ))

        # Config that is allowed:
        for k in ('html', 'linenumbers', 'forcemd', 'style', 'title'):
            arg = '--{}'.format(k)
            rawval = rawconfig.get(k, None)
            if rawval is not None:
                if argd[arg]:
                    # Do no override cmdline options.
                    debug('Option set: {:>15} == {}'.format(arg, argd[arg]))
                else:
                    argd[arg] = rawval
                    debug('Config set: {:>15} == {}'.format(arg, argd[arg]))
        # Argd has been overridden
        return argd


    def print_err(*args, **kwargs):
        """ A wrapper for print() that uses stderr by default. """
        if kwargs.get('file', None) is None:
            kwargs['file'] = sys.stderr
        print(*args, **kwargs)


    def print_styles():
        """ Print all known pygments styles and return a success status code. """
        print('\n'.join((
            '\nStyle names:',
            '    {}'.format(
                '\n    '.join(sorted(styles.STYLE_MAP))
            )
        )))
        return 0


    def read_stdin():
        """ Read from stdin, print a message if it's a terminal. """
        if sys.stdin.isatty() and sys.stdout.isatty():
            print('\nReading from stdin until end of file (Ctrl + D)...\n')
        return sys.stdin.read()


    class ConfigError(ValueError):
        """ Raised when config can't be loaded due to parsing errors. """
        pass


    class InvalidArg(ConfigError):
        """ Raised when the user has used an invalid argument. """
        def __init__(self, msg=None):
            self.msg = msg or ''

        def __str__(self):
            if self.msg:
                return 'Invalid argument, {}'.format(self.msg)
            return 'Invalid argument!'


    if __name__ == '__main__':
        try:
            mainret = main(docopt(USAGESTR, version=VERSIONSTR))
        except ConfigError as ex:
            print_err(ex)
            mainret = 1
        except (EOFError, KeyboardInterrupt):
            print_err('\nUser cancelled.\n', file=sys.stderr)
            mainret = 2
        except BrokenPipeError:
            print_err(
                '\nBroken pipe, input/output was interrupted.\n',
                file=sys.stderr)
            mainret = 3
        except EnvironmentError as ex:
            if ex.strerror and ex.filename:
                print_err(
                    '\n{x.strerror}: {x.filename}'.format(x=ex)
                )
            else:
                print_err('\n{}'.format(ex))
            mainret = 1
        sys.exit(mainret)
```