

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:

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

Command line help:

```
1 Usage:
2   codepdf -h | -S | -v
3   codepdf [FILE...] [-f] [-H] [-l] [-o file] [-s style] [-t title] [-D]
4
5 Options:
6   FILE                      : File names to convert, or - for stdin.
7                             : If no names are given, stdin is used.
8   -D,--debug                : Print some debug info while running.
9   -f,--forcemd              : Highlight markdown syntax, instead of
10                             : converting to HTML.
11   -h,--help                 : Show this help message.
12   -H,--html                 : Output in HTML instead of PDF.
13                             : Using .htm or .html as the output file
14                             : extension will automatically set this flag.
15   -l,--linenumbers          : Use line numbers.
16   -o file,--out file        : Output file name.
17                             : Default: <input_basename>.pdf
18   -s name,--style name      : Pygments style name to use for code files.
19                             : Default: default
20   -S,--styles               : Print all known pygments styles.
21   -t title,--title title    : Title for the PDF.
22                             : Default: <input_filename>
23   -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 your home dir (~), the current working dir (\$PWD), or next to codepdf.py.

The directories are tried in that order.

Single line javascript comments are acceptable.

Example config (codepdf.json):

```
1 {
2   // Syntax highlight markdown files instead of converting to HTML.
3   "forcemd": false,
4   // Use HTML output instead of PDF.
5   "html": false,
6   // Use line numbers for source files and readme code sections.
7   "linenumbers": false,
```

```
8 // Pygments style to use.
9 "style": "default",
10 // Title for the output PDF/HTML.
11 "title": null
12 }
```

Examples:

[example.html](#) is an HTML file that was created by running:

```
1 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:

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

requirements.txt

```
1 Colr>=0.2.5
2 docopt>=0.6.2
3 Markdown>=2.6.6
4 pdfkit>=0.5.0
5 Pygments>=2.1.3
```

codepdf.py

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 """ codepdf.py
5     Convert code/text files to pdf.
6     -Christopher Welborn 06-13-2016
7 """
8 # print function just to say "don't use python 2."
9 from __future__ import print_function
10 import inspect
11 import json
12 import os
13 import sys
14 try:
15     from contextlib import suppress
16 except ImportError as ex:
17     print('Error importing contextlib.suppress: {}'.format(ex))
18     if sys.version_info.major < 3:
19         # Better message than 'cannot import name suppress'
20         print(
21             '\n'.join((
22                 '\nCodePDF only works with Python 3+.',
23                 '\nCurrent python version:\n {}'.format(
24                     sys.version.replace('\n', '\n ')
25                 )),
26             ),
27             file=sys.stderr
28         )
29     sys.exit(1)
30
31 # Third-party libs.
32 try:
33     from colr import (
34         auto_disable as colr_auto_disable,
35         Colr as C
36     )
37     from docopt import docopt
38     from markdown import markdown
39     from markdown.extensions.codehilite import CodeHiliteExtension
40     from markdown.extensions.fenced_code import FencedCodeExtension
41     from markdown.extensions.sane_lists import SaneListExtension
42     from pdfkit import from_string as pdf_from_string
43     from pygments import highlight, lexers, formatters, styles
44     from pygments.util import ClassNotFound
45 except ImportError as ex:
46     print(
47         '\n'.join((
48             'Failed to import {pname}, you may need to install it:',
49             
```

```

50         'pip install {exc.name}',
51         'Original error:',
52         {exc.msg}'
53     ).format(
54         pname=exc.name.title(),
55         exc=exc
56     ),
57     file=sys.stderr
58 )
59 sys.exit(1)
60
61 # Disable colors when piping output.
62 colr_auto_disable()
63
64 NAME = 'CodePDF'
65 VERSION = '0.0.5'
66 VERSIONSTR = '{} v. {}'.format(NAME, VERSION)
67 SCRIPT = os.path.split(os.path.abspath(sys.argv[0]))[1]
68 SCRIPTDIR = os.path.abspath(sys.path[0])
69
70 # Global debug flag, set with --debug.
71 DEBUG = False
72 # File name to trigger reading from stdin.
73 STDIN_NAME = '-'
74 # Default pygments style.
75 DEFAULT_STYLE = 'default'
76 # Default pygments lexer, when it can't be detected.
77 DEFAULT_LEXER = 'text'
78 # Class name for each file's div.
79 DIV_CLASS = 'highlight'
80
81 USAGESTR = """{versionstr}
82 Usage:
83     {script} -h | -S | -v
84     {script} [FILE...] [-f] [-H] [-l] [-n] [-o file]
85             [-s style] [-t title] [-D]
86
87 Options:
88     FILE
89         : File names to convert, or {stdin} for stdin.
90         : If no names are given, stdin is used.
91     -D,--debug
92         : Print some debug info while running.
93     -f,--forcemd
94         : Highlight markdown syntax, instead of
95         : converting to HTML.
96     -h,--help
97         : Show this help message.
98     -H,--html
99         : Output in HTML instead of PDF.
100        : Using .htm or .html as the output file
101        : extension will automatically set this flag.
102     -l,--linenums
103         : Use line numbers.
104     -n,--noconfig
105         : Ignore config file settings.
106     -o file,--out file
107         : Output file name.
108         : Default: <input_basename>.pdf
109     -s name,--style name
110         : Pygments style name to use for code files.
111         : Default: {default_style}
112     -S,--styles
113         : Print all known pygments styles.
114     -t title,--title title
115         : Title for the PDF.
116         : Default: <input_filename>
117     -v,--version
118         : Show version.
119 """
120 .format(
121     default_style=DEFAULT_STYLE,
122     script=SCRIPT,
123     stdin=STDIN_NAME,
124     versionstr=VERSIONSTR
125 )
126
127 def main(argd):
128     """ Main entry point, expects doctopt arg dict as argd. """
129     global DEBUG
130     DEBUG = argd['--debug']
131     # Load user config, if applicable.
132     argd = load_config(argd)
133
134     if argd['--styles']:
135         return print_styles()
136
137     filenames = argd['FILE'] or [STDIN_NAME]
138     html_mode = argd['--html']
139     outname = get_output_name(
140         filenames,
141         output_name=argd['--out'],
142         html_mode=html_mode,
143     )
144
145     # Check for user-provided .html output file.
146     if not html_mode:
147         html_mode = outname.lower().endswith(('.htm', '.html'))
148
149     success = convert_files(
150         argd['FILE'] or [STDIN_NAME],
151         argd['--out'] or get_output_name(filenames),
152         stylename=argd['--style'],
153         linenos=argd['--linenums'],

```

```

141         title=argd['--title'],
142         force_highlight=argd['--forcemd'],
143         html_mode=html_mode,
144     )
145     if success:
146         print(outname)
147         return 0
148     return 1
149
150
151 def build_html(body, styles=None, title=None):
152     """ Try to build a somewhat-sane html page from a body and style-defs. """
153     if not styles:
154         styles = ['body {font-family: sans-serif;}']
155     else:
156         styles = list(styles)
157         styles.insert(0, 'body {font-family: sans-serif;}')
158
159     styles.append('\n'.join((
160         'hr {',
161         'border-style: hidden;',
162         'height: 2px;',
163         'background: #f1f1f1;',
164         'margin-top: 25px;',
165         '}',
166     )))
167     return '\n'.join((
168         '<html>',
169         '<head>',
170         '<title>{}</title>'.format(title or ''),
171         '<style type="text/css">',
172         '\n'.join(styles),
173         '</style>',
174         '</head>',
175         '<body>',
176         body,
177         '</body>',
178         '</html>'
179     ))
180
181
182 def convert_files(
183     filenames, outputname,
184     stylename=None, linenos=False,
185     title=None, force_highlight=False, html_mode=False):
186     """ Convert all files into a single PDF. """
187     stylename = stylename or DEFAULT_STYLE
188     debug(
189         '\n'.join((
190             'Converting files:\n    {}'.format(
191                 '\n    '.join(os.path.split(s)[-1] for s in filenames)
192             ),
193             'Output file: {outfile}',
194             '    Forced: {forced}',
195             '    LineNos: {linenos}',
196             '    Style: {style}',
197             '    Title: {title}',
198         )).format(
199             outfile=outputname,
200             forced=force_highlight,
201             linenos=linenos,
202             style=stylename,
203             title=title,
204         )
205     )
206     htmlcontent = []
207     styledefs = []
208     for i, filename in enumerate(filenames):
209         titletext = title or os.path.split(filename)[-1]
210         if titletext in (STDIN_NAME,):
211             titletext = 'stdin'
212         formatter = get_formatter(
213             stylename=stylename,
214             linenos=linenos,
215             title=titletext,
216         )
217         if not styledefs:
218             styledefs.append(formatter.get_style_defs())
219         htmlcontent.append(
220             convert_to_html_div(
221                 filename,
222                 formatter,
223                 stylename=stylename,
224                 linenos=linenos,
225                 force_highlight=force_highlight
226             )
227         )
228     allcontent = build_html(
229         '<hr class="nv">'.join(htmlcontent),
230         styles=styledefs,
231         title=titletext

```

```

232 )
233 if html_mode:
234     debug('Writing HTML to file...')
235     with open(outputname, 'w') as f:
236         f.write(allcontent)
237     return True
238
239 debug('Converting to PDF...')
240 return pdf_from_string(
241     allcontent,
242     outputname,
243     options={'--title': titletext, '--quiet': ''}
244 )
245
246
247 def convert_highlight(filename, formatter):
248     """ Highlight a file with pygments, and return the resulting HTML div. """
249     displayname, content = get_file_content(filename)
250     lexer = get_file_lexer(filename, content)
251     debug('Highlighting: {}'.format(displayname))
252     linkid = get_elem_id(displayname)
253     return '\n'.join((
254         '<div class="file">',
255         get_permalink_html(linkid),
256         '<h2 id="{0}" style="display: inline-block">{0}</h2>'.format(
257             linkid,
258             displayname
259         ),
260         '<div class="{0}">'.format(DIV_CLASS),
261         highlight(content, lexer, formatter),
262         '</div>',
263     ))
264
265
266
267 def convert_markdown(filename, stylename=None, linenos=False):
268     """ Convert a markdown file to an HTML div, and return the result. """
269     displayname, content = get_file_content(filename)
270     stylename = stylename.lower() if stylename else DEFAULT_STYLE
271     debug('Converting MD: {}'.format(displayname))
272     highlighter = CodeHiliteExtension(
273         pygments_style=stylename,
274         linenums=linenos,
275         noclasses=True,
276         css_class='highlight',
277     )
278     return '\n'.join((
279         '<div class="markdown">',
280         markdown(
281             content,
282             output_format='html5',
283             extensions=[
284                 highlighter,
285                 FencedCodeExtension(),
286                 SaneListExtension(),
287             ]
288         ),
289         '</div>'
290     ))
291
292
293 def convert_to_html_div(
294     filename, formatter,
295     stylename=None, linenos=False, force_highlight=False):
296     """ Convert a file to an html div.
297     The conversion method depends on the file extension.
298     build_html() should be used with the content returned here.
299     """
300     if (not force_highlight) and filename.endswith(('.md', '.markdown')):
301         return convert_markdown(
302             filename,
303             stylename=stylename,
304             linenos=linenos
305         )
306     return convert_highlight(filename, formatter)
307
308
309 def debug(*args, **kwargs):
310     """ Print a message only if DEBUG is truthy. """
311     if not (DEBUG and args):
312         return None
313
314     # Include parent class name when given.
315     parent = kwargs.get('parent', None)
316     with suppress(KeyError):
317         kwargs.pop('parent')
318
319     # Go back more than once when given.
320     backlevel = kwargs.get('back', 1)
321     with suppress(KeyError):
322         kwargs.pop('back')

```

```

323 frame = inspect.currentframe()
324 # Go back a number of frames (usually 1).
325 while backlevel > 0:
326     frame = frame.f_back
327     backlevel -= 1
328 fname = os.path.split(frame.f_code.co_filename)[-1]
329 lineno = frame.f_lineno
330 if parent:
331     func = '{}.{}'.format(parent.__class__.__name__, frame.f_code.co_name)
332 else:
333     func = frame.f_code.co_name
334
335 lineinfo = '{}:{}'.format(
336     C(fname, 'yellow'),
337     C(str(lineno).ljust(4), 'blue'),
338     C().join(C(func, 'magenta'), '()').ljust(20)
339 )
340 # Patch args to stay compatible with print().
341 pargs = list(C(a, 'green').str() for a in args)
342 pargs[0] = ''.join((lineinfo, pargs[0]))
343 print(*pargs, **kwargs)
344
345
346 def get_elem_id(s):
347     """ Transform a file name or text into a slug, usable for an element id.
348     Removes non alpha-numeric characters, replaces spaces with -.
349     """
350     return '-'.join(
351         ''.join(c for c in word if c.isalnum())
352         for word in s.split()
353     ).lower()
354
355
356 def get_file_content(filename):
357     """ Returns a tuple of (display_name, content), handling stdin if
358     STDIN_NAME is used.
359     """
360     if filename in (STDIN_NAME,):
361         return 'stdin', read_stdin()
362
363     with open(filename, 'r') as f:
364         content = f.read()
365     return os.path.split(filename)[-1], content
366
367
368 def get_file_lexer(filename, content):
369     """ Try to get a lexer by file extension, guess by content if that fails.
370     """
371     try:
372         # Pygments sometimes returns a weird lexer for .txt files.
373         if filename.lower().endswith('.txt'):
374             lexer = lexers.get_lexer_by_name('text')
375             debug('Lexer forced by extension: {:>20} -> {}'.format(
376                 lexer.name,
377                 filename,
378             ))
379         else:
380             lexer = lexers.get_lexer_for_filename(filename)
381             debug('Lexer chosen by file name: {:>20} -> {}'.format(
382                 lexer.name,
383                 filename,
384             ))
385     except ClassNotFound:
386         try:
387             # Guess by content.
388             lexer = lexers.guess_lexer(content)
389             debug('Lexer guessed by content: {:>20} -> {}'.format(
390                 lexer.name,
391                 filename,
392             ))
393         except ClassNotFound:
394             # Fall back to default lexer.
395             lexer = lexers.get_lexer_by_name(DEFAULT_LEXER)
396             debug('Lexer set to default: {:>20} -> {}'.format(
397                 lexer.name,
398                 filename,
399             ))
400     return lexer
401
402
403 def get_formatter(stylename=None, linenos=False, title=None, full=False):
404     """ Get an HTMLFormatter from pygments. """
405     stylename = stylename.lower() if stylename else DEFAULT_STYLE
406     try:
407         formatter = formatters.HtmlFormatter(
408             cssclass=DIV_CLASS,
409             linenos='inline' if linenos is True else linenos,
410             style=stylename,
411             full=full,
412             title=title
413

```

```

414     )
415     except ClassNotFound:
416         raise InvalidArg(
417             '\n'.join((
418                 'Unknown style name: {style}',
419                 'Run `{script} --styles` to see a list of style names.',
420             )).format(
421                 style=stylename,
422                 script=SCRIPT,
423             )
424         )
425     return formatter
426
427
428 def get_permalink_html(linkid):
429     """ Return HTML needed to build a permalink link/icon for a header. """
430     svg = """
431     <svg style="vertical-align: middle; display: inline;"
432     height="16" version="1.1" viewBox="0 0 16 16" width="16">
433     <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
434     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
435     0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 2.5S13.98 12 13 12H9c-.98
436     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
437     7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z">
438     </path></svg>
439     """
440     return '\n'.join((
441         '<a href="#{linkid}" style="text-decoration: none;">'.format(linkid=linkid),
442         svg,
443         '</a>'
444     ))
445
446
447 def get_output_name(filename, output_name=None, html_mode=False):
448     """ Determine output file name to use when the user hasn't given one. """
449     if output_name:
450         # Short-circuit auto-name-detection.
451         return output_name
452
453     inputname = filename
454     if inputname == '-':
455         inputname = 'stdin'
456     parentdir, basename = os.path.split(inputname)
457     if not parentdir:
458         parentdir = os.getcwd()
459     return '{name}{ext}'.format(
460         name=os.path.join(parentdir, os.path.splitext(basename)[0]),
461         ext='.html' if html_mode else '.pdf'
462     )
463
464
465 def load_config(argd):
466     """ Load config from ~/.codepdf.json or ./codepdf.json.
467     Override argd values with user config.
468     """
469     if argd['--noconfig']:
470         debug('Config ignored.')
471         return argd
472
473     filename = 'codepdf.json'
474     for trydir in (os.getcwd(), os.path.expanduser('~'), SCRIPTDIR):
475         filepath = os.path.join(trydir, filename)
476         if os.path.exists(filepath):
477             debug('Found config file: {}'.format(filepath))
478             break
479     else:
480         debug('No config file found: {}'.format(filename))
481         return argd
482
483     with open(filepath, 'r') as f:
484         content = ''.join(l for l in f if not l.strip().startswith('//'))
485
486     try:
487         rawconfig = json.loads(content)
488     except ValueError as ex:
489         raise ConfigError('Unable to load config from: {} \n {}'.format(
490             filepath,
491             ex
492         ))
493
494     # Config that is allowed:
495     for k in ('html', 'linenumbers', 'forcemd', 'style', 'title'):
496         arg = '--{}'.format(k)
497         rawval = rawconfig.get(k, None)
498         if rawval is not None:
499             if argd[arg]:
500                 # Do no override cmdline options.
501                 debug('Option set: {>15} == {}'.format(arg, argd[arg]))
502             else:
503                 argd[arg] = rawval
504                 debug('Config set: {>15} == {}'.format(arg, argd[arg]))

```

```

505     # Argv has been overridden
506     return argv
507
508
509 def print_err(*args, **kwargs):
510     """ A wrapper for print() that uses stderr by default. """
511     if kwargs.get('file', None) is None:
512         kwargs['file'] = sys.stderr
513     print(*args, **kwargs)
514
515
516 def print_styles():
517     """ Print all known pygments styles and return a success status code. """
518     print('\n'.join((
519         '\nStyle names:',
520         '{}'.format(
521             '\n'.join(sorted(styles.STYLE_MAP))
522         )
523     )))
524     return 0
525
526
527 def read_stdin():
528     """ Read from stdin, print a message if it's a terminal. """
529     if sys.stdin.isatty() and sys.stdout.isatty():
530         print('\nReading from stdin until end of file (Ctrl + D)...')
531     return sys.stdin.read()
532
533
534 class ConfigError(ValueError):
535     """ Raised when config can't be loaded due to parsing errors. """
536     pass
537
538
539 class InvalidArg(ConfigError):
540     """ Raised when the user has used an invalid argument. """
541     def __init__(self, msg=None):
542         self.msg = msg or ''
543
544     def __str__(self):
545         if self.msg:
546             return 'Invalid argument, {}'.format(self.msg)
547         return 'Invalid argument!'
548
549
550 if __name__ == '__main__':
551     try:
552         mainret = main(docopt(USAGESTR, version=VERSIONSTR))
553     except ConfigError as ex:
554         print_err(ex)
555         mainret = 1
556     except (EOFError, KeyboardInterrupt):
557         print_err('\nUser cancelled.\n', file=sys.stderr)
558         mainret = 2
559     except BrokenPipeError:
560         print_err(
561             '\nBroken pipe, input/output was interrupted.\n',
562             file=sys.stderr)
563         mainret = 3
564     except EnvironmentError as ex:
565         if ex.strerror and ex.filename:
566             print_err(
567                 '\n{x.strerror}: {x.filename}'.format(x=ex)
568             )
569         else:
570             print_err('\n{}'.format(ex))
571         mainret = 1
572     sys.exit(mainret)

```