Project: Source2PDF

File: build_pdf.sh

Created By: kellpossible, Date: 2013-10-08

```
#!/bin/sh
#currently hidden folders with fullstop prefix are automatically excluded
#anyway, but this is just an example of how you might exclude it using regex
python2 Source2Pdf.py --ext py sh css --exclude .*[\\.]git.* \
-o Source2Pdf.pdf --project-name Source2Pdf --style default --user-name kellpossible \
--line-numbers
```

File: Source2Pdf.py

Created By: kellpossible, Date: 2013-10-07

```
#!/usr/bin/env python2

"""
Source2Pdf.py
This file is part of Source2PDF

Copyright (C) 2013 - Luke Frisken

Source2PDF is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

Source2PDF is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with Source2PDF. If not, see <http://www.gnu.org/licenses/>.
"""


import cStringIO as StringIO
import os, os.path, sys, datetime, pwd, re
import argparse
from subprocess import Popen, PIPE, STDOUT
import ho.pisa as pisa

from pygments import highlight
from pygments.lexers import PythonLexer
from pygments.formatters import HtmlFormatter
from pygments.lexers import guess_lexer_for_filename
from pygments.styles import STYLE_MAP

def get_current_directory():
        return os.getcwd()

def get_source_directory():
        file_path = os.path.realpath(sys.argv[0])
        return os.path.dirname(file_path) + "/"

class ProjectDocument(object):
        """This Class Represents the document of the overall project"""
```

```
45          def __init__(self, path, args):
46                  self.args = args
47                  self.file_documents = []
48                  self.path = path
49                  self.get_project_stats()
50                  self.main_re = re.compile(".*main[.].*", re.IGNORECASE)
51
52          def append(self, document):
53                  #TODO: make main_re part of the options
54                  #insert any document with path containing main at the start
55                  match = self.main_re.match(document.path)
56                  if match:
57                          self.file_documents.insert(0, document)
58                  else:
59                          #append any others to the end ;)
60                          self.file_documents.append(document)
61
62          def get_project_stats(self):
63                  self.cloc = 0
64                  self.name = os.path.basename(self.path)
65
66          def to_html(self):
67                  """Convert document to html"""
68                  html = "<html>"
69                  html += """
70  <style>
71  @page {
72    margin: 1cm;
73    margin-bottom: 2.5cm;
74    @frame footer {
75      -pdf-frame-content: footerContent;
76      bottom: 1cm;
77      margin-left: 1cm;
78      margin-right: 1cm;
79      height: 1cm;
80    }
81    @frame footer {
82      -pdf-frame-content: headerContent;
83      top: 0.5cm;
84      margin-left: 17.5cm;
85      margin-right: 1cm;
86      height: 1cm;
87    }
88  }
89  font {
90          font-size: 140%;
91  }"""
92                  #add pygments styles to stylesheet
93                  style = "bw"
94                  if self.args.style:
95                          style = self.args.style
96                  html += HtmlFormatter(style=style).get_style_defs('.highlight')
97
98
99                  html +="\n</style>"
100                  html +="""
101  <div id="footerContent">
102     page <pdf:pagenumber />
103  </div>
104  """
105                  html += """<div id="headerContent" style="margin-left: -2cm">
106     <b>Project: {0}</b>
107  </div>""".format(self.name)
108                  html += """<h1 style="font-size:300%">Project: {0}</h1>
109                  """.format(self.name)
```

```
110                 for d in self.file_documents:
111                         html += d.to_html()
112
113                 html += "</style>"
114                 return html
115
116         def to_pdf_file(self, filepath):
117                 """Convert document to pdf file"""
118                 f = open(filepath, 'w')
119                 print("starting")
120                 html = self.to_html()
121                 #f.write(html)
122                 try:
123                         encoded_html = html.encode("ISO-8859-1")
124                 except:
125                         changed_html = self.validate_encoding(html)
126                         encoded_html = html
127                 pdf = pisa.CreatePDF(StringIO.StringIO(encoded_html), f)
128                 print(pdf.err)
129                 #if not pdf.err:
130                         #pisa.startViewer(filepath)
131                 f.close()
132
133         def validate_encoding(self, text):
134                 #re.sub('[\xe2]', '', text)
135                 text.translate(None, '\xe2')
136
137
138         def to_html_file(self, filepath):
139                 f = open(filepath, 'w')
140                 html = self.to_html()
141                 f.write(html)
142                 f.close()
143
144
145 class FileDocument(object):
146         """This class represents a document of a single file in
147         the project"""
148         userinfo = {}
149         def __init__(self, path, args):
150                 self.args = args
151                 self.path = path #path to file associated with document
152                 self.get_user_info()
153                 self.get_file_stats()
154
155         def get_user_info(self):
156                 """Get user info about this file from the filesystem"""
157                 for ui in pwd.getpwall():
158                         self.userinfo[ui[2]] = ui
159
160         def get_file_stats(self):
161                 """get file stats about this file from the filesystem"""
162                 stats = os.stat(self.path)
163                 if self.args.username:
164                         self.username = self.args.username
165                 else:
166                         user = self.userinfo[stats.st_uid]
167                         self.username = user.pw_gecos #use real username
168                         #self.username = user.pw_name #use the short username
169
170                 self.modifytime = datetime.date.fromtimestamp(stats.st_mtime)
171
172         def to_html(self):
173                 """Convert file document to html"""
174                 source_dir = get_source_directory()
```

```
175                 css_path = source_dir + "printing.css"
176
177                 fin = open(self.path, 'r')
178                 code = fin.read()
179
180                 #cmd = "source-highlight -n --style-css-file {0} --tab 2 -f html -i {1}".format(css_path, self.path)
181                 #p = Popen(cmd.split(), shell=False, stdout=PIPE)
182
183                 file_path_name = os.path.relpath(self.path, get_current_directory())
184                 html_string = """<h1>File: {0}</h1>
185                 <h3>Created By: {1}, Date: {2}</h3>
186                 """.format(file_path_name,
187                                         self.username,
188                                         str(self.modifytime))
189
190                 lexer = guess_lexer_for_filename('test.py', code, stripall=True)
191
192                 linenos = False
193                 if self.args.linenumbers == True:
194                         linenos = 'inline'
195
196                 formatter = HtmlFormatter(style='bw', linenos=linenos)
197                 html_string += highlight(code, lexer, formatter)
198
199                 fin.close()
200                 return html_string
201
202         def to_latex(self):
203                 """return string of latex"""
204                 pass
205
206 class Searcher(object):
207         """This class is used for searching the filsystem for
208         the relevent files to include in the project"""
209         code_extensions = ['c', 'h',
210                                                 'cpp', 'hpp',
211                                                 'cs',
212                                                 'go',
213                                                 'java',
214                                                 'py']
215
216         def __init__(self, args):
217                 self.args = args
218                 if args.extensions == None:
219                         self.extensions = self.code_extensions #auto extensions
220                 else:
221                         self.extensions = args.extensions
222
223                 self.build_re_extensions()
224
225                 self.re_exclusions = []
226                 if args.exclusions:
227                         self.exclusions = args.exclusions
228                         self.build_re_exclusions()
229
230                 if args.files == None:
231                         self.SEARCH_ARGS = False
232                 else:
233                         self.SEARCH_ARGS = True
234                         self.arg_filenames = args.files
235
236                 self.documents = []
237
238
239         def build_re_extensions(self):
```

```
240                 """Build regular expression for searching
241                 file extensions"""
242                 extensions_list = ""
243                 for ext in self.extensions:
244                         extensions_list += "{0}|".format(ext)
245 
246                 extensions_list = extensions_list[:-1] #cut final bar
247                 re_string = ".*[.]({0})$".format(extensions_list)
248                 print(re_string)
249                 self.extension_re = re.compile(re_string)
250 
251         def build_re_exclusions(self):
252                 """Build regular expression for excluding files"""
253                 for ex in self.exclusions:
254                         re_ex = re.compile(ex)
255                         self.re_exclusions.append(re_ex)
256 
257         def test_exclude(self, f):
258                 for re_ex in self.re_exclusions:
259                         if re_ex.match(f):
260                                 return True
261 
262                 return False
263 
264         def search(self):
265                 self.documents.append(ProjectDocument(get_current_directory(), self.args))
266                 if not self.SEARCH_ARGS:
267                         self.searchAuto()
268                 else:
269                         self.searchArgs()
270 
271         def searchAuto(self):
272                 current_dir = os.getcwd()
273                 for root, dirs, filenames in os.walk(current_dir):
274                         for f in filenames:
275                                 fpath = os.path.join(root, f)
276                                 #print(root + "/" + f)
277                                 match = self.extension_re.match(f)
278                                 if match:
279                                         if self.test_exclude(fpath):
280                                                 print("excluded: {0}".format(fpath))
281                                                 continue
282                                         self.documents[0].append(FileDocument(fpath, self.args))
283 
284         def searchArgs(self):
285                 for fname in self.arg_filenames:
286                         root = get_current_directory()
287                         #print(root, fname)
288                         path = os.path.join(root, fname)
289                         #print(path)
290                         self.documents[0].append(FileDocument(path, self.args))
291 
292         def handle_file_search(self, root, f):
293                 pass
294 
295 #extensions = extension_s.split(";")
296 
297 if __name__ == '__main__':
298         parser = argparse.ArgumentParser(description='Convert code to PDF files')
299         parser.add_argument('-e','--ext', metavar='Ext', type=str, dest='extensions', nargs='*',
300                 help='the letters of the file extensions')
301         parser.add_argument('-x','--exclude', metavar='Regex', type=str, dest='exclusions', nargs='*',
302                 help='each element of exclude is a regex to be excluded')
303         parser.add_argument('-o', metavar='File', type=str, dest='outfile', nargs='?',
304                 help='name of output file')
```

```
305         parser.add_argument('-u', '--user-name', type=str, dest='username', nargs='?',
306                 help='set custom user name')
307         parser.add_argument('-n', '--project-name', type=str, dest='project_name', nargs='?',
308                 help='set custom project name')
309         parser.add_argument('--style', type=str, dest='style', nargs='?',
310                 choices=STYLE_MAP.keys(),
311                 help='set pygments style')
312         parser.add_argument('-l', '--line-numbers', dest='linenumbers', action='store_true',
313                 help='use line numbers')
314         parser.add_argument('-i', metavar='File', type=str, dest='files', nargs='*',
315                 help='file names to be converted')
316 
317         args = parser.parse_args()
318         outfile = args.outfile
319         s = Searcher(args)
320         s.search()
321         doc = s.documents[0]
322 
323         if not outfile:
324                 outfile = "render.pdf"
325 
326 
327         if outfile.split(".")[1] == "pdf":
328                 #doc.to_html_file("render.html")
329                 doc.to_pdf_file(outfile)
330         else:
331                 doc.to_html_file(outfile)
```

File: styles/github.css

Created By: kellpossible, Date: 2013-10-07

```
 1 .hll { background-color: #ffffcc }
2 .c { color: #999988; font-style: italic } /* Comment */
3 .err { color: #a61717; background-color: #e3d2d2 } /* Error */
4 .k { color: #000000; font-weight: bold } /* Keyword */
5 .o { color: #000000; font-weight: bold } /* Operator */
6 .cm { color: #999988; font-style: italic } /* Comment.Multiline */
7 .cp { color: #999999; font-weight: bold; font-style: italic } /* Comment.Preproc */
8 .c1 { color: #999988; font-style: italic } /* Comment.Single */
9 .cs { color: #999999; font-weight: bold; font-style: italic } /* Comment.Special */
10 .gd { color: #000000; background-color: #ffdddd } /* Generic.Deleted */
11 .ge { color: #000000; font-style: italic } /* Generic.Emph */
12 .gr { color: #aa0000 } /* Generic.Error */
13 .gh { color: #999999 } /* Generic.Heading */
14 .gi { color: #000000; background-color: #ddffdd } /* Generic.Inserted */
15 .go { color: #888888 } /* Generic.Output */
16 .gp { color: #555555 } /* Generic.Prompt */
17 .gs { font-weight: bold } /* Generic.Strong */
18 .gu { color: #aaaaaa } /* Generic.Subheading */
19 .gt { color: #aa0000 } /* Generic.Traceback */
20 .kc { color: #000000; font-weight: bold } /* Keyword.Constant */
21 .kd { color: #000000; font-weight: bold } /* Keyword.Declaration */
22 .kn { color: #000000; font-weight: bold } /* Keyword.Namespace */
23 .kp { color: #000000; font-weight: bold } /* Keyword.Pseudo */
24 .kr { color: #000000; font-weight: bold } /* Keyword.Reserved */
25 .kt { color: #445588; font-weight: bold } /* Keyword.Type */
26 .m { color: #009999 } /* Literal.Number */
27 .s { color: #d01040 } /* Literal.String */
28 .na { color: #008080 } /* Name.Attribute */
29 .nb { color: #0086B3 } /* Name.Builtin */
30 .nc { color: #445588; font-weight: bold } /* Name.Class */
31 .no { color: #008080 } /* Name.Constant */
32 .nd { color: #3c5d5d; font-weight: bold } /* Name.Decorator */
33 .ni { color: #800080 } /* Name.Entity */
```

```
.ne { color: #990000; font-weight: bold } /* Name.Exception */
.nf { color: #990000; font-weight: bold } /* Name.Function */
.nl { color: #990000; font-weight: bold } /* Name.Label */
.nn { color: #555555 } /* Name.Namespace */
.nt { color: #000080 } /* Name.Tag */
.nv { color: #008080 } /* Name.Variable */
.ow { color: #000000; font-weight: bold } /* Operator.Word */
.w { color: #bbbbbb } /* Text.Whitespace */
.mf { color: #009999 } /* Literal.Number.Float */
.mh { color: #009999 } /* Literal.Number.Hex */
.mi { color: #009999 } /* Literal.Number.Integer */
.mo { color: #009999 } /* Literal.Number.Oct */
.sb { color: #d01040 } /* Literal.String.Backtick */
.sc { color: #d01040 } /* Literal.String.Char */
.sd { color: #d01040 } /* Literal.String.Doc */
.s2 { color: #d01040 } /* Literal.String.Double */
.se { color: #d01040 } /* Literal.String.Escape */
.sh { color: #d01040 } /* Literal.String.Heredoc */
.si { color: #d01040 } /* Literal.String.Interpol */
.sx { color: #d01040 } /* Literal.String.Other */
.sr { color: #009926 } /* Literal.String.Regex */
.s1 { color: #d01040 } /* Literal.String.Single */
.ss { color: #990073 } /* Literal.String.Symbol */
.bp { color: #999999 } /* Name.Builtin.Pseudo */
.vc { color: #008080 } /* Name.Variable.Class */
.vg { color: #008080 } /* Name.Variable.Global */
.vi { color: #008080 } /* Name.Variable.Instance */
.il { color: #009999 } /* Literal.Number.Integer.Long */
```