

# repo-to-pdf

## Contents

/bin

[|- index.js](#)

[|- html.js](#)

[|- pandoc.js](#)

[|- render.js](#)

[|- repo.js](#)

[|- utils.js](#)

src/bin/index.js

[to top](#)

```
#!/usr/bin/env node
```

```
let inputFolder, outputFile, renderer,  
calibrePath, baseUrl, protocol
```

```
const fs = require('fs')  
const path = require('path')  
const os = require('os')
```

```
const program = require('commander')
```

```
const { getSizeInByte } = require('../utils')  
const { generateEbook } = require('../html')
```

```
const version =  
require('../package.json').version  
const PDF_SIZE = getSizeInByte(10) // 10 Mb
```

```
program  
  .version('repo-to-pdf ' + version)  
  .usage('<input> [output] [options]')  
  .arguments('<input> [output] [options]')  
  .option('-d, --device <platform>', 'device  
[desktop(default)|mobile|tablet]', 'desktop')  
  .option('-t, --title [name]', 'title')  
  .option('-w, --whitelist [wlist]', 'file format  
white list, split by ,')  
  .option('-s, --size [size]', 'pdf file size  
limit, in Mb')  
  .option('-r, --renderer <engine>', 'use chrome  
or calibre to render pdf', 'node')  
  .option('-f, --format <ext>', 'output format,  
pdf|mobi|epub', 'pdf')  
  .option('-c, --calibre [path]', 'path to  
calibre')  
  .option('-b, --baseUrl [url]', 'base url of  
html folder. By default file:// is used.')
```

```
.action(function(input, output) {  
    inputFolder = input  
    outputFile = output  
})
```

```
program.parse(process.argv)
```

```
const title = program.title || inputFolder  
const device = program.device  
const pdf_size = program.size ?  
getSizeInByte(program.size) : PDF_SIZE  
const format = program.format  
const white_list = program.whitelist  
renderer = program.renderer  
calibrePath = program.calibre  
if (program.baseUrl) {  
    protocol = ''  
    baseUrl = program.baseUrl  
} else {  
    protocol = os.name === 'windows' ? 'file:////' :  
'file:/'  
    baseUrl = path.resolve(__dirname,  
'../../html5bp')  
}
```

```
if (format !== 'pdf' && renderer === 'node') {  
    console.log(`Try to create ${format}, use  
renderer Calibre.`)  
    renderer = 'calibre'  
}
```

```
// check calibre path  
const calibrePaths =  
['/Applications/calibre.app/Contents/MacOS/ebook-  
convert', '/usr/bin/ebook-convert']  
if (calibrePath) {  
    calibrePaths.unshift(calibrePath)  
}
```

```
let i = 0
for (; i < calibrePaths.length; i++) {
  if (fs.existsSync(calibrePaths[i])) {
    calibrePath = calibrePaths[i]
    break
  }
}
if (i === calibrePaths.length && ['mobi',
'epub'].includes(format)) {
  console.log('Calibre ebook-convert not found,
make sure you pass it by --calibre
/path/to/ebook-convert.')
  return
}

generateEbook(inputFolder, outputFile, title, {
  renderer,
  calibrePath,
  pdf_size,
  white_list,
  format,
  device,
  baseUrl,
  protocol,
})
```

src/html.js

[to top](#)

```
const path = require('path')
const fs = require('fs')

const { Remarkable } = require('remarkable')
const hljs = require('highlight.js')

const RepoBook = require('./repo')
const { sequenceRenderEbook } =
  require('./render')
const { getFileName, getFileNameExt } =
  require('./utils')

function getRemarkableParser() {
  return new Remarkable({
    breaks: true,
    highlight: function(str, lang) {
      if (lang && hljs.getLanguage(lang)) {
        try {
          return hljs.highlight(lang, str).value
        } catch (err) {}
      }

      try {
        return hljs.highlightAuto(str).value
      } catch (err) {}

      return ''
    },
  }).use(function(remarkable) {
    remarkable.renderer.rules.heading_open =
function(tokens, idx) {
  return '<h' + tokens[idx].hLevel + ' id=' +
tokens[idx + 1].content + ' anchor=true>'
}
  })
}
```

```
// => './path/file-1.html'
function getHTMLFiles(mdString, repoBook,
options) {
  const { pdf_size, white_list, device, baseUrl,
protocol, renderer, outputFileName, inputFolder }
= options
  const opts = {
    cssPath: {
      desktop: '/css/github-min.css',
      tablet: '/css/github-min-tablet.css',
      mobile: '/css/github-min-mobile.css',
    },
    highlightCssPath: '/css/vs.css',
    relaxedCSS: {
      desktop: '',
      tablet: `@page {
        size: 8in 14in;
        -relaxed-page-width: 8in;
        -relaxed-page-height: 14in;
        margin: 0;
      }`,
      mobile: `@page {
        size: 6in 10in;
        -relaxed-page-width: 6in;
        -relaxed-page-height: 10in;
        margin: 0;
      }`,
    },
  },
}

  const HTMLFileNameWithExt =
getFileNameExt(outputFileName, 'html') ||
getFileName(inputFolder) + '.html'

  let outputFile = path.resolve(process.cwd(),
HTMLFileNameWithExt)

  const mdParser = getRemarkableParser()

  const mdHtml = `<article class="markdown-
```

```

body">` + mdParser.render(mdString) +
`</article>`
    const indexHtmlPath = path.join(__dirname,
'../html5bp', 'index.html')
    const htmlString = fs
        .readFileSync(indexHtmlPath, 'utf-8')
        // TODO: this sits before content replacing,
to prevent replacing baseUrl in content text
        .replace(/\{\{baseUrl\}\}/g, protocol +
baseUrl)
        .replace('{{cssPath}}', protocol + baseUrl +
opts.cssPath[device])
        .replace('{{highlightPath}}', protocol +
baseUrl + opts.highlightCssPath)
        .replace('{{relaxedCSS}}',
opts.relaxedCSS[device])
        .replace('{{content}}', mdHtml)

    if (!repoBook.hasSingleFile()) {
        outputFile = outputFile.replace('.html', '-')
+ repoBook.currentPart() + '.html')
    }
    fs.writeFileSync(outputFile, htmlString)
    return outputFile
}

```

```

function generateEbook(inputFolder, outputFile,
title, options = { renderer: 'node' }) {
    const { pdf_size, white_list, renderer } =
options
    const repoBook = new RepoBook(inputFolder,
title, pdf_size, white_list)

    const defaultOutputFileName =
getFileName(inputFolder) + '.pdf'
    const outputFileName = outputFile ||
defaultOutputFileName

```

```
options.outputFileName = outputFileName
options.inputFolder = inputFolder
options.outputFile = outputFile

const outputFiles = []
while (repoBook.hasNextPart()) {
  const mdString = repoBook.render()
  let outputFile = null;
  if (renderer === 'node') {
    outputFile = getHTMLFiles(mdString,
repoBook, options)
  } else if (renderer === 'calibre') {
    outputFile = getHTMLFiles(mdString,
repoBook, options)
  }
  if (!outputFile) {
    console.log('generation failed, unknown
exception.')
    break
  }
  outputFiles.push(outputFile)
}
sequenceRenderEbook(outputFiles, options)
}

module.exports = { generateEbook }
```

src/pandoc.js

[to top](#)



```

function getMarkdownVars() {
  return [
    '___',
    'title: context book',
    'papersize: a4',
    'margin-left: 2',
    'margin-right: 2',
    'margin-top: 2',
    'margin-bottom: 2',
    `header-includes: |
      <style>
        body {
          font-family: sans-serif;
          font-size: 48px;
        }
      </style>
      <link rel="stylesheet"
href="/Users/chenjosh/projects/repo-to-
pdf/html5bp/css/normalize.css">
      <link rel="stylesheet"
href="/Users/chenjosh/projects/repo-to-
pdf/html5bp/css/main.css">
      <link rel="stylesheet"
href="/Users/chenjosh/projects/repo-to-
pdf/html5bp/css/github-min.css">
      <script src="/Users/chenjosh/projects/repo-
to-pdf/html5bp/js/vendor/modernizr-2.6.2.min.js">
</script>`,
    '___'
  ].join('\n')
}

// => './path/file-1.md'
function getMarkdownFiles(mdString, repoBook,
options) {
  const { outputFileName, inputFolder } = options

```

```

    const markdownFileNameWithExt =
    getFileNameExt(outputFileName, 'md') ||
    getFileName(inputFolder) + '.md'
    let outputFile = path.resolve(process.cwd(),
    markdownFileNameWithExt)

    if (!repoBook.hasSingleFile()) {
        outputFile = outputFile.replace('.md', '-' +
        repoBook.currentPart() + '.md')
    }

    const markdownVars = getMarkdownVars()
    fs.writeFileSync(outputFile, [markdownVars,
    mdString].join('\n'))
    return outputFile
}

// pandoc: ['pandoc', [
//   '--from', 'markdown',
//   '--to', 'pdf',
//   '-t', 'html5',
//   // '--css', './css/github-min.css',
//   // '--data-dir',
//   '/Users/chenjosh/projects/repo-to-pdf/html5bp',
//   '-V', 'margin-left=2',
//   '-V', 'margin-right=2',
//   '-V', 'margin-top=2',
//   '-V', 'margin-bottom=2',
//   '-o', formatFile, docFile
// ]]

```

src/render.js

[to top](#)

```
const path = require('path')
const fs = require('fs')
const { spawnSync } = require('child_process')

const { getFileNameExt } = require('./utils')

let startTs

function removeRelaxedjsTempFiles(outputFileName)
{
  const prefix =
outputFileName.split('.').reverse().slice(1).reverse()

  const html = spawnSync('rm',
[`${prefix}.html`])
  const htm = spawnSync('rm',
[`${prefix}_temp.htm`])
}

function reportPerformance(outputFileName,
startTs) {
  const ts = (Date.now() - startTs) / 1000
  console.log(`${outputFileName} created in ${ts}
seconds.`)
}

function sequenceRenderEbook(docFiles, options, i
= 0) {
  const { outputFileName, renderer, calibrePath,
format } = options

  if (i === 0) {
    startTs = Date.now()
  }
  if (i >= docFiles.length) {
    if (renderer === 'node') {
      removeRelaxedjsTempFiles(outputFileName)
```

```
    }
    reportPerformance(outputFileName, startTs)
    return
  }
  const docFile = path.resolve(process.cwd(),
docFiles[i])
  const formatFile = getFileNameExt(docFile,
format)

  const formatArgs = {
    pdf: [
      '--pdf-add-toc',
      '--paper-size',
      'a4',
      '--pdf-default-font-size',
      '12',
      '--pdf-mono-font-size',
      '12',
      '--pdf-page-margin-left',
      '2',
      '--pdf-page-margin-right',
      '2',
      '--pdf-page-margin-top',
      '2',
      '--pdf-page-margin-bottom',
      '2',
      '--page-breaks-before',
      '/',
    ],
    mobi: ['--mobi-toc-at-start', '--output-
profile', 'kindle_dx'],
    epub: ['--epub-inline-toc', '--output-
profile', 'ipad3', '--flow-size', '1000'],
  }

  const args = {
    node: ['npx', ['relaxed', docFile, '--build-
once', '--no-sandbox']],
```

```
    calibre: [calibrePath, [docFile,
formatFile].concat(formatArgs[format])]
  }

  if (renderer === 'calibre') {
    if (!fs.existsSync(calibrePath)) {
      console.log(`Calibre is not available.`)
    }
  }

  const cmd = args[renderer]
  const res = spawnSync(cmd[0], cmd[1])
  if (res.error) {
    console.log(`Some error happened when
creating ${formatFile}.`)
    return
  }

  if (!fs.existsSync(formatFile)) {
    console.log(`${docFiles[i]} was not
rendered.`)
  }

  sequenceRenderEbook(docFiles, options, i + 1)
}

module.exports = { sequenceRenderEbook }
```

[src/repo.js](#)

[to top](#)

```
const path = require('path')
const fs = require('fs')
const hljs = require('highlight.js')
const { getFileName, getCleanFilename } =
  require('./utils')

class RepoBook {
  constructor(dir, title, pdf_size, white_list) {
    this.title = title
    this.pdf_size = pdf_size

    this.blackList = ['node_modules', 'vendor']
    this.whiteList = white_list ?
white_list.split(',') : null

    this.aliases = {}
    this.byteOffset = 0
    this.fileOffset = 0
    this.partOffset = 0
    this.done = false
    this.dir = dir

    this.files = this.readDir(dir)
    this.registerLanguages()
  }

  hasNextPart() {
    return this.done === false
  }

  hasSingleFile() {
    return this.partOffset === 0 // effective
after at least calling render() once
  }

  currentPart() {
    return this.partOffset
  }
}
```

```

}

registerLanguage(name, language) {
  const lang = language(hljs)
  if (lang && lang.aliases) {
    name = name.split('.')[0]
    this.aliases[name] = name
    lang.aliases.map(alias => {
      if (this.whiteList) {
        if (this.whiteList.indexOf(alias) > -1)
        {
          this.aliases[alias] = name.split('.')[0]
        }
      } else {
        this.aliases[alias] = name.split('.')[0]
      }
      return null
    })
  }
}

registerLanguages() {
  const listPath =
path.join(path.dirname(require.resolve('highlight.js')),
'languages')
  fs.readdirSync(listPath).map(f => {
    this.registerLanguage(f,
require(path.join(listPath, f)))
    return null
  })
}

readDir(dir, allFiles = [], level = 0) {
  const files = fs
    .readdirSync(dir)
    .map(f => path.join(dir, f))

```

```

        .filter(f => fs.lstatSync(f).size / 1000 <
2000) // smaller than 2m
        .map(f => [f, level])

    level > 0 ? allFiles.push([dir, level, true])
: null // push folder name

    files.map(pair => {
        const f = pair[0]
        const blackListHit =
this.blackList.filter(e => !!f.match(e)).length >
0
        if (!fs.lstatSync(f).isDirectory()) {
            allFiles.push([f, level + 1, false])
        } else {
            path.basename(f)[0] !== '.' && // ignore
hidden folders
            blackListHit === false &&
            this.readDir(f, allFiles, level + 1)
        }
        return null
    })
    return allFiles
}

renderIndex(files) {
    return files
        .filter(f => {
            const fileName = getFileName(f[0])
            const ext =
path.extname(fileName).slice(1)
            const isFolder =
fs.lstatSync(f[0]).isDirectory()
            return fileName[0] !== '.' && (ext in
this.aliases || isFolder)
        })
        .map(f => {
            const indexName = getCleanFilename(f[0],

```



```
this.dir, f[1]),  
    anchorName = getCleanFilename(f[0],  
this.dir),  
    left_pad = f[2] ?  
'&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;' .repeat(f[1]) : '',  
    h_level = '####',  
    list_style = f[2] ? '' :  
'&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;' .repeat(f[1]) + '|-'  
    return f[2] ? `${h_level} ${left_pad}  
${list_style} /${indexName}` : `${h_level}  
${left_pad}[${list_style} ${indexName}]  
(${anchorName})`  
    })  
    .join('\n')  
}
```

```
render() {  
  const files = this.files  
  const contents = []  
  let i = this.fileOffset  
  
  for (; i < files.length; i++) {  
    const file = files[i][0]  
  
    const fileName = getFileName(file)  
    if (fs.statSync(file).isDirectory()) {  
      continue  
    }  
  
    const ext = path.extname(fileName).slice(1)  
  
    if (ext.length === 0) {  
      continue  
    }  
  
    if (fileName[0] === '.') {  
      continue  
    }  

```

```

const lang = this.aliases[ext]
if (lang) {
  let data = fs.readFileSync(file)
  if (ext === 'md') {
    data = `#### ${getCleanFilename(file,
this.dir)} \n[to top](#Contents)` + '\n' + data +
'\n'
  } else {
    data = `#### ${getCleanFilename(file,
this.dir)} \n[to top](#Contents)` + '\n` `` ' +
lang + '\n' + data + '\n` ``\n'
  }
  contents.push(data)

  this.byteOffset += data.length * 2
  if (this.byteOffset > this.pdf_size) {
    // if more than one part
    const title = `# ${this.title}
(${++this.partOffset})\n\n\n\n`

    let toc = `## Contents\n`
    const index =
this.renderIndex(files.slice(this.fileOffset, i +
1))

    toc += index
    contents.unshift(title, toc)

    if (i === this.fileOffset) {
      // when single file exceeds size
limit
      this.fileOffset++
    } else {
      this.fileOffset = i
    }
    this.byteOffset = 0

    return contents.join('\n')
  }
}

```

```

    }
  }
}

if (contents.length === 0) {
  return null
}

// if one part
const title = this.partOffset
  ? `# ${this.title}
  (${++this.partOffset})\n\n\n\n` // the last part
  : `# ${this.title} \n\n\n\n` // single pdf

let toc = '## Contents\n'
const index =
this.renderIndex(files.slice(this.fileOffset, i +
1))
toc += index
contents.unshift(title, toc)

this.fileOffset = files.length // should
return null next round
this.done = true

return contents.join('\n')
}
}

module.exports = RepoBook

```

src/utils.js

[to top](#)

```
const path = require('path')

function getSizeInByte(mb) {
  return mb * 0.8 * 1000 * 1000
}

// '../' => 'untitled'
// './' => 'untitled'
// './path' => 'path'
function getFileName(fpath) {
  const base = path.basename(fpath)
  return base[0] === '.' ? 'untitled' : base
}

function getCleanFilename(filename, folder, depth
= 0) {
  filename = filename.replace(folder, '')
  if (depth > 0) {
    return filename
      .split('/')
      .slice(depth)
      .join('/')
  } else {
    return filename
  }
}

// 'file.random_extension' => 'file.ext'
function getFileNameExt(fileName, ext = 'pdf') {
  return fileName.replace(/\.[0-9a-zA-Z]+$/,
    `.${ext}`)
}

module.exports = { getSizeInByte, getFileName,
getCleanFilename, getFileNameExt }
```