

目次

GitBook によるドキュメント作成	1.1
カスタマイズ	1.1.1
プラグイン	1.1.2
テーマ	1.1.2.1
フォント	1.1.2.2
SNSボタン	1.1.2.3
ハイライト	1.1.2.4
コードのコピーボタン	1.1.2.5
ソースコードの挿入	1.1.2.6
GitBookで公開リンクを削除	1.1.2.7
アンカー	1.1.2.8
目次の追加	1.1.2.9
ページのトップに戻るボタン	1.1.2.10
見出しに番号を自動でつける	1.1.2.11
サイドメニューの折りたたみ	1.1.2.12
警告などのメッセージ	1.1.2.13
注釈	1.1.2.14
絵文字	1.1.2.15
TODOリスト	1.1.2.16
UML	1.1.2.17
Mermaid	1.1.2.18
グラフ	1.1.2.19
数式	1.1.2.20
Tips	1.1.3
PDF	1.1.4
参考	1.1.5

1 GitBook によるドキュメント作成

GitBook は Markdown形式のファイルからドキュメントを作成するツールです。HTML形式、PDF形式、EPUB形式、MOBI形式を作成することができます。

ここでは HTML および PDF 形式のみ確認しています。

過去に以下のような記事を書きました。

- [MkDocsによるドキュメント作成](#)
- [DoxygenによるMarkdown形式からのドキュメント作成](#)

それぞれメリット・デメリットがあります。今回の GitBook は最終的に PDF 出力が出来て、そこそこ体裁が整っているのが最大の魅力だと思います。

1.1 gitbook-cli

GitBook はクラウドサービスですが、GitBookの機能をコマンドラインから操作できるツール `gitbook-cli` があります。これを使用してオフラインでも GitBook を使ってドキュメントを作成します。

1.2 環境構築

Miniconda (Anaconda) を使って環境を構築します。まずは `gitbook` という仮想環境を作成します。

```
conda create -n gitbook
conda activate gitbook
```

`gitbook-cli` は Nodejs アプリなので、Nodejs をインストールします。

```
conda install nodejs
```

`npm` を使って `gitbook-cli` をインストールします。

```
$ npm install gitbook-cli -g
$ gitbook --version
CLI version: 2.3.2
GitBook version: 3.2.3
```

1.3 プロジェクトの作成

`init` コマンドで空のプロジェクトを作成できます。試しに `example` というプロジェクトを作成します。

```
$ mkdir example
$ cd example
$ gitbook init
warn: no summary file in this book
info: create README.md
info: create SUMMARY.md
info: initialization is finished
```

`README.md` と `SUMMARY.md` ファイルが作成されます。`README.md` はトップページになります。

```
// README.md

# Introduction
```

SUMMARY.md には文書の構成を記述します。これはサイドバーに表示される構造になります。

```
// SUMMARY.md

# Summary

* [Introduction](README.md)
```

1.4 ビルド

HTML形式で出力するには `build` コマンドを使います。

```
$ gitbook build
info: 7 plugins are installed
info: 6 explicitly listed
info: loading plugin "highlight"... OK
info: loading plugin "search"... OK
info: loading plugin "lunr"... OK
info: loading plugin "sharing"... OK
info: loading plugin "fontsettings"... OK
info: loading plugin "theme-default"... OK
info: found 1 pages
info: found 0 asset files
info: >> generation finished with success in 0.6s !
```

ビルドしたデータは `_book` フォルダに作成されます。このフォルダの中をデプロイすることで公開することができます。



また、`serve` コマンドを使うことで、ローカルでサーバーを起動して表示確認することができます。

```
$ gitbook serve
Live reload server started on port: 35729
Press CTRL+C to quit ...

info: 7 plugins are installed
info: loading plugin "livereload"... OK
info: loading plugin "highlight"... OK
info: loading plugin "search"... OK
info: loading plugin "lunr"... OK
info: loading plugin "sharing"... OK
info: loading plugin "fontsettings"... OK
info: loading plugin "theme-default"... OK
info: found 1 pages
info: found 0 asset files
info: >> generation finished with success in 0.6s !

Starting server ...
Serving book on http://localhost:4000
```

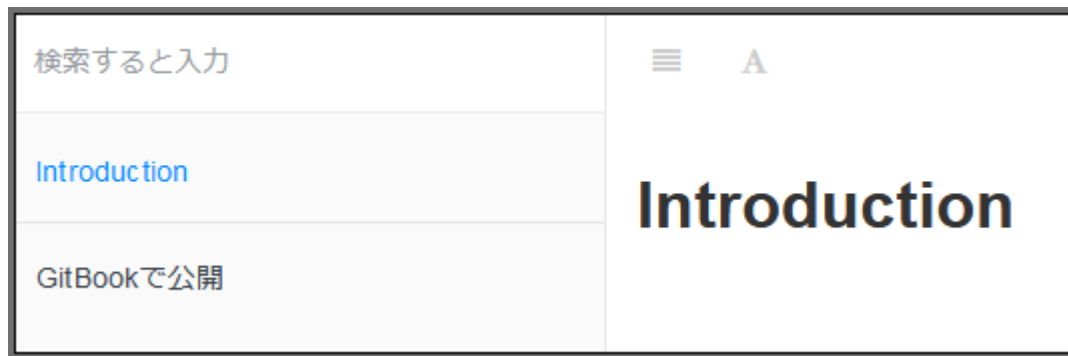
ブラウザで `http://localhost:4000` を開いて確認します。

Type to search	 
Introduction	<h1>Introduction</h1>
Published with GitBook	

2 カスタマイズ

各種設定は `book.json` ファイルで指定します。次は日本語設定したシンプルな設定です。

```
{
  "language": "ja"
}
```



タイトルや概要は `title` , `description` で指定します。

```
{
  "title": "GitBook Tutorial",
  "description": "This is a gitbook example"
}
```

指定しなければ、`README.md` の最初のヘッダ、パラグラフが使われます。 `root` を使用するとドキュメントの検索フォルダを変更することができます。

```
{
  "root": "src"
}
```

`styles` で拡張スタイルシートを指定することができます。

```
{
  "styles": {
    "website": "styles/website.css",
    "ebook": "styles/ebook.css",
    "pdf": "styles/pdf.css",
    "mobi": "styles/mobi.css",
    "epub": "styles/epub.css"
  }
}
```

3 プラグイン

GitBook はプラグインを導入してカスタマイズすることができます。空のプロジェクトを作成した時点でいくつかのプラグインが自動で有効になります。

- livereload
- highlight
- search
- lunr
- sharing
- fontsettings
- theme-default

プラグインの追加は `plugins` で設定します。

```
{  
  "plugins": [...]  
}
```

また、プラグインの設定は `pluginsConfig` で設定します。

```
{  
  "plugins": [...],  
  "pluginsConfig": {...}  
}
```

`plugins` に追加したプラグインは `gitbook install` でインストールします。

```
gitbook install
```

様々なプラグインがありますので、個人的にオススメなものをピックアップしていきます。

4 テーマ

デフォルトのテーマは `theme-default` です。他には `theme-api` , `theme-official` がいい感じです。テーマはそのテーマのプラグインを有効にすることで反映されます。まずは、デフォルトのテーマは次のようなものです。



上部にあるアイコンから、フォントの大きさやフォントのセリフ・サンセリフ、そして、テーマカラーの切り替えが出来ます。



Sepia



Night



次に `theme-api` です。

```
{  
  "plugins": ["theme-api"]  
}
```




theme-api は上の画像だと違いがわかりづらいですが、特徴的な機能として2列レイアウトがあります。



また、赤丸で囲んだアイコンをクリックすると通常表示と2列表示を切り替えることができます。このサンプルは以下のようにになっています。

```
{% method %}
## 環境構築 {#env}

Miniconda (Anaconda) を使って環境を構築します。
まずは `GitBook` という仮想環境を作成します。

{% sample lang="shell" %}
\ \ \

conda create -n gitbook
conda activate gitbook
\ \ \

{% endmethod %}

{% method %}
## Node.js のインストール {#nodejs}
GitBook-cli は Nodejs アプリなので, Nodejs をインストールします。

{% sample lang="shell" %}
\ \ \

conda install nodejs
\ \ \

{% endmethod %}

{% method %}
## gitbook-cli のインストール {#gitbookcli}

npm を使って GitBook-cli をインストールします。

{% sample lang="shell" %}
\ \ \

$ npm install gitbook-cli -g
$ gitbook --version
CLI version: 2.3.2
GitBook version: 3.2.3
\ \ \

{% endmethod %}
```

この `theme-api` にはダークモードがあります。これはプラグインの設定で指定します。

```
{
  "plugins": ["theme-api"],
  "pluginsConfig": {
    "theme-api": {
      "theme": "dark"
    }
  }
}
```

検索すると入力

GitBook によるドキュメント作成

theme-api

GitBookで公開

theme-api sample

環境構築

Miniconda (Anaconda) を使って環境を構築します。まずは `GitBook` という仮想環境を作成します。

```
conda create -n gitbook
conda activate gitbook
```

Node.js のインストール

GitBook-cli は Nodejs アプリなので、Nodejs をインストールします。

```
conda install nodejs
```

gitbook-cli のインストール

npm を使って GitBook-cli をインストールします。

```
$ npm install gitbook-cli -g
$ gitbook --version
CLI version: 2.3.2
GitBook version: 3.2.3
```

`theme-default` ではコンテンツの幅が標準で `800px` になっていますが、`theme-api` の場合、ウィンドウの幅に合わせて伸縮します。

次のテーマは、`theme-official` です。

```
{
  "plugins": ["theme-official"]
}
```

GitBook Tutorial

検索すると入力

GitBook によるドキュメント作成

GitBook によるドキュメント作成

GitBook は Markdown形式のファイルからドキュメントを作成するツールです。HTML形式、PDF形式、EPUB形式、MOBI形式を作成することができます。

GitBook-cli

GitBook はクラウドサービスですが、GitBookの機能をコマンドラインから操作できるツール `GitBook-cli` があります。これを使用します。

環境構築

Miniconda (Anaconda) を使って環境を構築します。まずは `GitBook` という仮想環境を作成します。

```
conda create -n gitbook
conda activate gitbook
```

GitBook-cli は Nodejs アプリなので、Nodejs をインストールします。

```
conda install nodejs
```

npm を使って GitBook-cli をインストールします。

```
$ npm install gitbook-cli -g
```

11

5 フォント

CSS で指定します。初期状態では次のようになっています。



例えば、本文に `Noto Serif JP` , `Noto Sans JP` , 見出しレベル1に `Roboto Slab` , `pre`, `code` には `Roboto Mono` を指定した `website.css` は次のようになります。

```
@import url(https://fonts.googleapis.com/css?family=Noto+Sans+JP|Noto+Serif+JP|Roboto+Mono|Roboto+Slab&display=block);

.book.font-family-0 {
  font-family: "Noto Serif JP", "メイリオ", serif;
}

.book.font-family-1 {
  font-family: "Noto Sans JP", "メイリオ", sans-serif;
}

.markdown-section h1,
.markdown-section h2,
.markdown-section h3,
.markdown-section h4,
.markdown-section h5 {
  font-family: "Roboto Slab", "Noto Sans JP", sans-serif;
}

.markdown-section pre,
.markdown-section code {
  font-family: "Roboto Mono", Consolas, "Courier New", courier, monospace;
}
```

`.book.font-family-0` はセリフ系フォントを指定します。



`.book.font-family-1` はサンセリフ系を設定します。



見出しの装飾を次のように指定すると

```
@import url(https://fonts.googleapis.com/css?family=Noto+Sans+JP|Noto+Serif+JP|Roboto+Mono|Roboto+Slab&disp]

.book.font-family-0 {
  font-family: "Noto Serif JP", "メイリオ", serif;
}

.book.font-family-1 {
  font-family: "Noto Sans JP", "メイリオ", sans-serif;
}

.markdown-section h1,
.markdown-section h2,
.markdown-section h3,
.markdown-section h4,
.markdown-section h5 {
  font-family: "Roboto Slab", "Noto Sans JP", sans-serif;
  font-weight: bold;
  color: rgb(0,113,188);
}

.markdown-section h1 {
  border-bottom: 1px solid #000;
}

.markdown-section h2 {
  border-bottom: 1px dotted #888;
}

.markdown-section pre,
.markdown-section code {
  font-family: "Roboto Mono", Consolas, "Courier New", courier, monospace;
}
```

検索すると入力

GitBook によるドキュメント作成

GitBookで公開

≡ A

🐦 f ↻

GitBook によるドキュメント作成

GitBook は Markdown形式のファイルからドキュメントを作成するツールです。HTML形式、PDF形式、EPUB形式、MOBI形式を作成することができます。

gitbook-cli

GitBook はクラウドサービスですが、GitBookの機能をコマンドラインから操作できるツール `gitbook-cli` があります。これを使用してオフラインでも GitBook を使ってドキュメントを作成します。

環境構築

Miniconda (Anaconda) を使って環境を構築します。まずは `gitbook` という仮想環境を作成します。

```
conda create -n gitbook
conda activate gitbook
```

6 SNSボタン

右上に表示される SNSボタンは `sharing` プラグインです。これは標準で有効になっています。



非表示にしたい場合はプラグインを無効にします。

```
{
  "plugins": ["-sharing"]
}
```

また、個別に指定したい場合は次のようになります。

```
{
  "pluginsConfig": {
    "sharing": [
      "facebook": true,
      "twitter": true,
      "google": false,
      "weibo": false,
      "instapaper": false,
      "vk": false,
      "all": [
        "facebook", "google", "twitter", "weibo", "instapaper"
      ]
    ]
  }
}
```

7 コードハイライト

コードのハイライト機能は標準で `highlight` が有効になっています。

```
#include <iostream>
int main() {
    std::cout << "Hello World!" << std::endl;
    return 0;
}
```

```
console.log('Hello World!');
```

7.1 prism

ハイライトのプラグインは他に `prism` があります。有効にする時は標準のハイライトプラグインを無効にしておきます。

```
{
  "plugins": ["prism", "-highlight"]
}
```

```
#include <iostream>
int main() {
    std::cout << "Hello World!" << std::endl;
    return 0;
}
```

```
console.log('Hello World!');
```

`prism` には以下のテーマが用意されています。

- Okaidia (prismjs/themes/prism-okaidia.css)
- Solarized Light (prismjs/themes/prism-solarizedlight.css)
- Tomorrow (prismjs/themes/prism-tomorrow.css)
- Dark (prismjs/themes/prism-dark.css)
- Coy (prismjs/themes/prism-coy.css)
- Funky (prismjs/themes/prism-funky.css)
- Twilight (prismjs/themes/prism-twilight.css)

テーマを指定するには次のようにします。


```
{
  "pluginsConfig": {
    "prism": {
      "css": [
        "prismjs/themes/prism-solarizedlight.css"
      ]
    }
  }
}
```

7.1.0.1 sunlight-highlighter

他のコードハイライトプラグインとして `sunlight-highlighter` がオススメです。これは行番号を表示することができます。

```
{
  "plugins": ["sunlight-highlighter", "-highlight"],
  "pluginsConfig": {
    "sunlightHighlighter": {
      "theme": "gitbook",
      "lineNumbers": true
    }
  }
}
```

```
1 #include <iostream>
2 int main() {
3     std::cout << "Hello World!" << std::endl;
4     return 0;
5 }
```

```
1 console.log('Hello World!');
```

テーマとして標準の `gitbook` の他に `light` , `dark` が用意されています。

```
1 #include <iostream>
2 int main() {
3     std::cout << "Hello World!" << std::endl;
4     return 0;
5 }
```

```
1 console.log('Hello World!');
```

```
1 #include <iostream>
2 int main() {
3     std::cout << "Hello World!" << std::endl;
4     return 0;
5 }
```

```
1 console.log('Hello World!');
```

8 コードのコピーボタン

コードをクリップボードにコピーするボタンを表示する `copy-code-button` プラグインがあります。

```
{  
  "plugins": ["copy-code-button"]  
}
```

```
#include <iostream>  
int main() {  
    std::cout << "Hello World!" << std::endl;  
    return 0;  
}
```

Copy

残念ながら、コードハイライトプラグイン `sunlight-highlighter` を合わせて使うと表示が崩れてしまいます。

9 ソースコードの挿入

ソースコードのファイルを挿入してシンタックスハイライトを適用する `include-codeblock` プラグインがあります.

```
{
  "plugins": ["include-codeblock"]
}
```

例えば次のような `src/main.cpp` ファイルがあります.

```
#include <iostream>
int main() {
  std::cout << "Hello World!" << std::endl;
  return 0;
}
```

これをドキュメントに挿入する場合は次のようになります.

```
``` c_cpp
#include <iostream>
int main() {
 std::cout << "Hello World!" << std::endl;
 return 0;
}
```

または

```
#include <iostream>
int main() {
 std::cout << "Hello World!" << std::endl;
 return 0;
}
```

```
``` c_cpp
#include <iostream>
int main() {
  std::cout << "Hello World!" << std::endl;
  return 0;
}
```

c++のソースコードを挿入すると言語は `c_cpp` となりますが, `prism` にはその言語に対応していません. そこで, エイリアスを設定することで対応することができます.

```
{
  "pluginsConfig": {
    "prism": {
      "lang": {
        "c_cpp": "cpp"
      }
    }
  }
}
```

`include-codeblock` ではテンプレートに `ace` を設定することができます。 `ace` はエディタープラグインです。これを使って `monokai` テーマを適用すると

```
{
  "plugins": ["include-codeblock", "ace"],
  "pluginsConfig": {
    "include-codeblock": {
      "template": "ace",
      "theme": "monokai"
    }
  }
}
```

```
1 #include <iostream>
2 int main() {
3     std::cout << "Hello World!" << std::endl;
4     return 0;
5 }
```

ただし、テンプレートを `ace` にした場合は `copy-code-button` プラグインの対象にならないようです。

`include-codeblock` には様々な機能があります。興味のある人は以下を参照してください。

- [gitbook-plugin-include-codeblock](#)

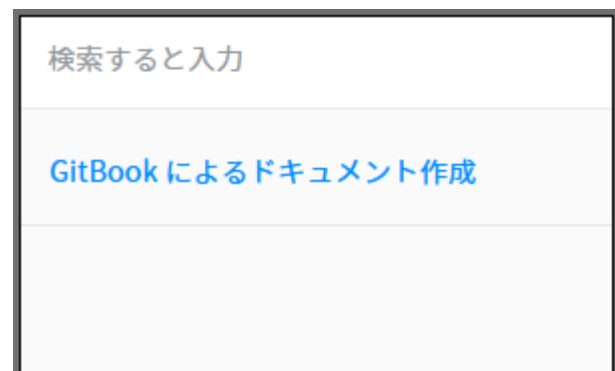
10 GitBookで公開リンクを削除

サイドバーにある `GitBookで公開` (Published with GitBook) を削除するには `hide-published-with` プラグインを使います.

```
{  
  "plugins": ["hide-published-with"]  
}
```



これがようになります.



11 アンカー

`anchors` を使うと各見出しにアンカーを自動で付けてくれます。

```
{  
  "plugins": ["anchors"]  
}
```



12 目次の追加

いくつかプラグインがあります。

12.1 navigator

右上に目次（折りたたみ可能）、右下にトップに戻るボタンが追加されます。

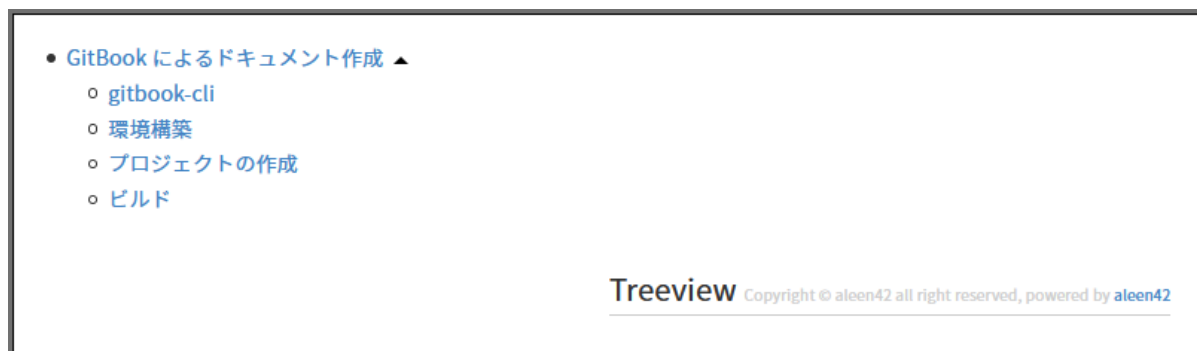
```
{  
  "plugins": ["navigator"]  
}
```



12.2 page-treeview

ページの上部に目次をツリー形式で追加されます。

```
{  
  "plugins": ["page-treeview"]  
}
```



右下にコピーライト表記が追加されます。これは設定で消すことができます。また、目次に追加する見出しの調整も出来ます。

```
{
  "pluginsConfig": {
    "page-treeview": {
      "copyright": "",
      "minHeaderCount": "2",
      "minHeaderDeep": "2"
    }
  }
}
```

12.3 atoc

`page-treeview` で著作権表記を消すと上部に隙間が空いてしまいます。代わりに `atoc` プラグインを使えば、この隙間がなくなりスッキリします。

```
{
  "plugins": ["atoc"]
}
```

ただし、`atoc` の場合は各ページに `<!-- toc -->` を追加しなければなりません。

12.4 intopic-toc

右上に目次を表示します。標準では見出しレベル2のものが表示されます。

```
{
  "plugins": ["intopic-toc"]
}
```

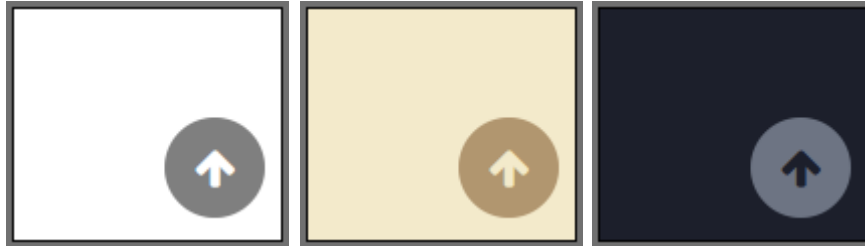


後述する `back-to-top-button` プラグインと組み合わせるといい感じです。

13 ページのトップに戻るボタン

`navigator` プラグインでも追加することは出来ませんが、外観がよい `back-to-top-button` プラグインもオススメです。このプラグインはデフォルトテーマの各テーマカラーに対応しています。

```
{  
  "plugins": ["back-to-top-button"]  
}
```



14 見出しに番号をつける

`numbered-headings-for-web-and-books` を使います.

```
{  
  "plugins": ["numbered-headings-for-web-and-books"]  
}
```

1. GitBook によるドキュメント作成

GitBook は Markdown 形式のファイルからドキュメントを作成するツールです. HTML 形式, PDF 形式, EPUB 形式, MOBI 形式を作成することができます.

1.1 gitbook-cli

GitBook はクラウドサービスですが, GitBook の機能をコマンドラインから操作できるツール `gitbook-cli` があります. これを使用してオフラインでも GitBook を使ってドキュメントを作成します.

1.2 環境構築

Miniconda (Anaconda) を使って環境を構築します. まずは `gitbook` という仮想環境を作成します.

```
conda create -n gitbook  
conda activate gitbook
```

Copy

15 サイドメニューの折りたたみ

`SUMMARY.md` に書かれているSUMMARYのツリーを折りたたみできるようにするプラグインがあります。

15.1 expand-active-chapter

まずは `expand-active-chapter` プラグインです。これはアクティブなツリーのみ展開します。

```
{  
  "plugins": ["expand-active-chapter"]  
}
```

GitBook によるドキュメント作成

カスタマイズ

プラグイン

テーマ

フォント

SNSボタン

ハイライト

コードのコピーボタン

ソースコードの挿入

15.2 collapsible-chapters

`collapsible-chapters` は折りたたみ可能なノードの戦闘にアイコンが付き、わかりやすくなります。

```
{  
  "plugins": ["collapsible-chapters"]  
}
```

▼ GitBook によるドキュメント作成

カスタマイズ

▼ プラグイン

テーマ

フォント

SNSボタン

ハイライト

コードのコピーボタン

ソースコードの挿入

16 警告などのメッセージ

`hints` プラグインを使います.

```
{  
  "plugins": ["hints"]  
}
```

`info` , `tip` , `danger` , `working` の4種類があります. 使い方は次のようになります.

```
{% hint style='info' %}  
info: this is a information  
{% endhint %}  
  
{% hint style='tip' %}  
tip: this is a tip  
{% endhint %}  
  
{% hint style='danger' %}  
danger: this is a danger message  
{% endhint %}  
  
{% hint style='working' %}  
working: this is working  
{% endhint %}
```

info: this is a information

tip: this is a tip

danger: this is a danger message

working: this is working

17 注釈

`footnote-string-to-number` を使うと便利です¹。文字列を数値に変換してくれます。

```
{  
  "plugins": ["footnote-string-to-number"]  
}
```

次のように使います。

```
This is a pen[^2].
```

```
[^2]: I like it.
```

¹. description here ↩

18 絵文字

advanced-emoji プラグインを使います。

```
{  
  "plugins": ["advanced-emoji"]  
}
```

次のように使います。

```
:smile:, :heart_eyes:, :blush:
```

😊, 😍, 🥰

絵文字は以下を参照したり検索エンジンで検索してみてください。

- [Complete List of Emoji](#)

19 TODOリスト

todo プラグインを使います.

```
{  
  "plugins": ["todo"]  
}
```

次のように使います.

```
- [x] apple  
- [x] orange  
- [ ] cherry  
- [ ] strawberry
```

- ☒ apple
- ☒ orange
- ☐ cherry
- ☐ strawberry

20 UML

uml プラグインを使います.

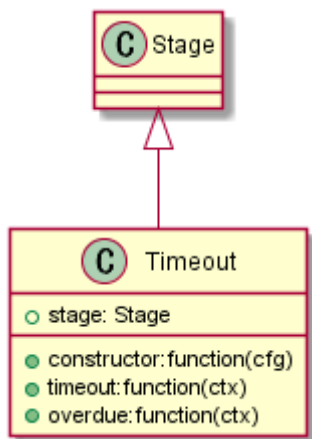
```
{  
  "plugins": "uml"  
}
```

次のように使います.

{% uml %} @startuml

```
Class Stage  
Class Timeout {  
  +constructor:function(cfg)  
  +timeout:function(ctx)  
  +overdue:function(ctx)  
  +stage: Stage  
}  
Stage <|-- Timeout
```

@enduml {% enduml %}



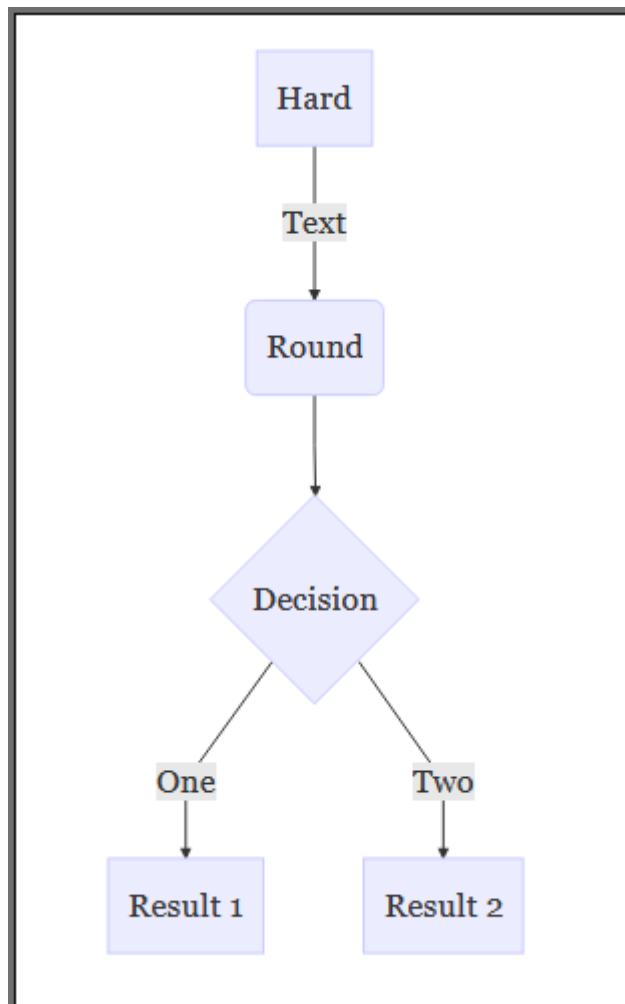
21 Mermaid

ダイアグラムやフローチャートなどの図を作成できる Mermaid のプラグインは `mermaid-gb3` を使います。

```
{  
  "plugins": ["mermaid-gb3"]  
}
```

次のように使います。

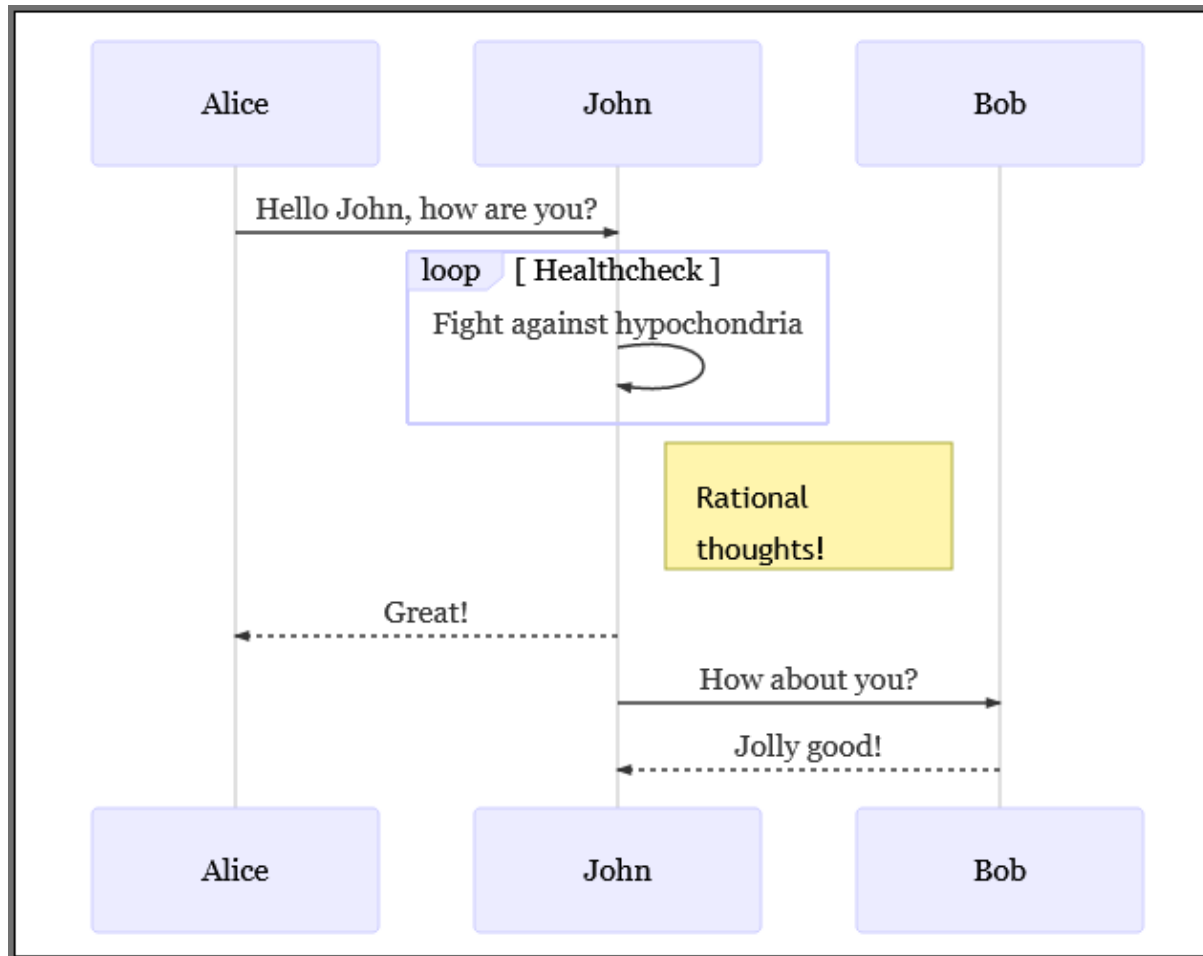
```
```mermaid  
graph TD
 A[Hard] -->|Text| B(Round)
 B --> C{Decision}
 C -->|One| D[Result 1]
 C -->|Two| E[Result 2]
```
```



```

```mermaid
sequenceDiagram
 Alice->>John: Hello John, how are you?
 loop Healthcheck
 John->>John: Fight against hypochondria
 end
 Note right of John: Rational thoughts!
 John-->>Alice: Great!
 John->>Bob: How about you?
 Bob-->>John: Jolly good!
```

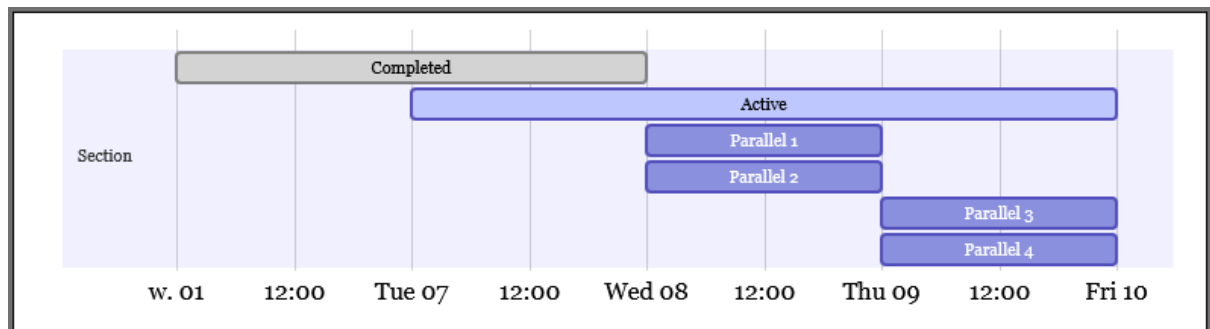
```



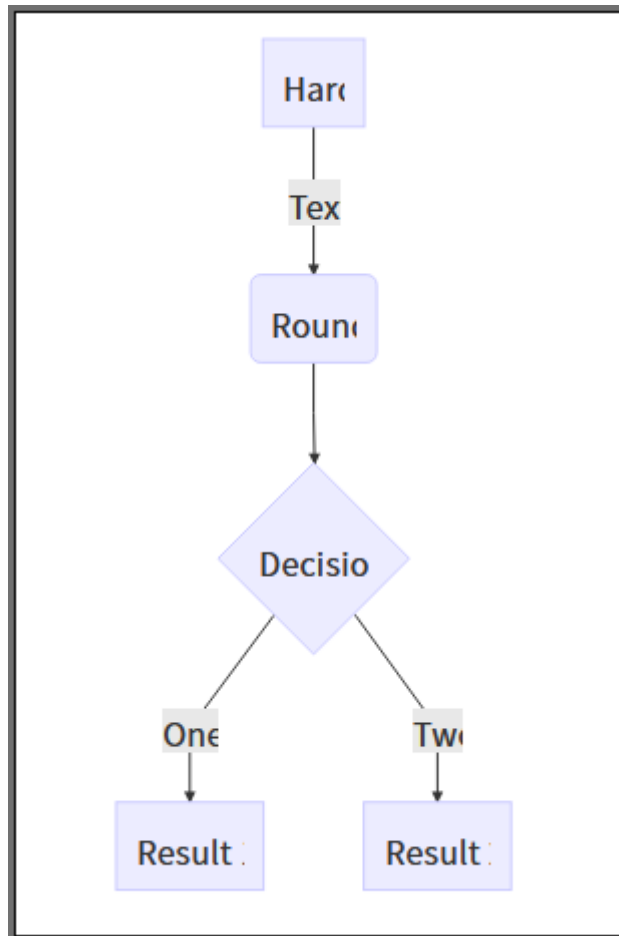
```

```mermaid
gantt
 section Section
 Completed :done, des1, 2014-01-06,2014-01-08
 Active :active, des2, 2014-01-07, 3d
 Parallel 1 : des3, after des1, 1d
 Parallel 2 : des4, after des1, 1d
 Parallel 3 : des5, after des3, 1d
 Parallel 4 : des6, after des4, 1d
```

```



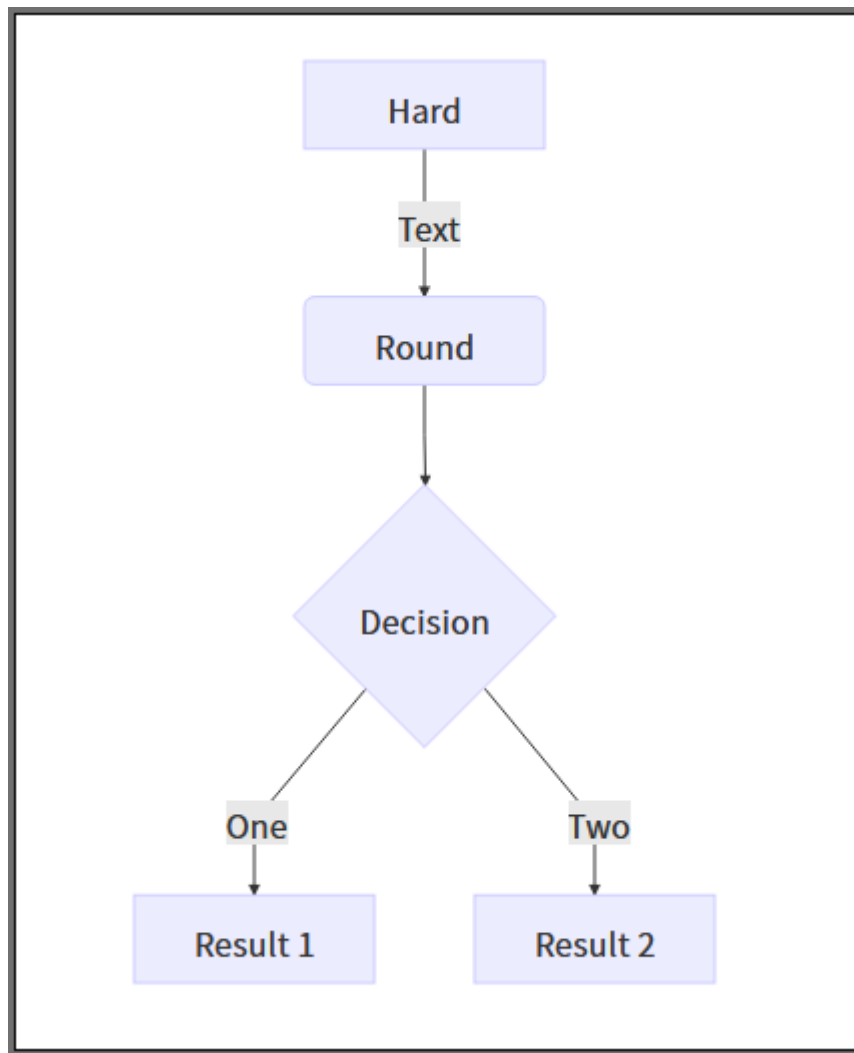
フォントを標準から変更したところ、フローチャートでは表示が一部切れてしまっていました。



根本的な解決方法はわかっていないのですが、例えばスタイルシートを次のように調整すると

```
g.label foreignObject {
  width: 100px;
  text-align: center;
}

g.label foreignObject span.edgeLabel {
  margin-left: -70px;
}
```



いい感じに見えますが、固定幅になっていますので図によって再調整しなければなりません。

22 グラフ

`graph` プラグインを使います.

```
{  
  "plugins": ["graph"]  
}
```

次のように使います.

```
{% graph %}  
  {  
    "title": "cos(2*PI*x/2)*(1+0.5cos(2*PI*x/100))",  
    "grid": true,  
    "xAxis": {  
      "label": "Sample",  
      "domain": [0, 300]  
    },  
    "yAxis": {  
      "label": "Amplitude",  
      "domain": [-1.5, 1.5]  
    },  
    "data": [  
      { "fn": "cos(2*PI*x/2)*(1+0.5cos(2*PI*x/100))" },  
      { "fn": "(1+0.5cos(2*PI*x/100))" }  
    ]  
  }  
{% endgraph %}
```

23 数式

数式を表示するには `mathjax` か `katex` プラグインを使います。 `Mathjax` は数式を表示するための Javascript ライブラリです。 `Katex` は `Mathjax` より高速に処理しますが、表現できる数式に制限があります。まずは `mathjax` プラグインを使う場合は次のようになります。

```
{
  "plugins": ["mathjax"],
  "pluginsConfig": {
    "mathjax": {
      "forceSVG": true,
      "version": "2.7.6"
    }
  }
}
```

SVG形式で出力します。また、`Mathjax`のバージョンがメジャーアップしているのを、一応バージョンを指定しています。次に `katex` の場合は次のようになります。

```
{
  "plugins": ["katex"]
}
```

使い方は両方とも同じです。

```
When  $a \neq 0$ ,
there are two solutions to  $(ax^2 + bx + c = 0)$ 
and they are  $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ .

When  $a \neq 0$ ,
there are two solutions to  $(ax^2 + bx + c = 0)$ 
and they are  $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ .
```

When $a \neq 0$, there are two solutions to $(ax^2 + bx + c = 0)$ and they are $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$.

When $a \neq 0$, there are two solutions to $(ax^2 + bx + c = 0)$ and they are $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$.

`mathjax` プラグインを使用している場合、次のようなエラーが出る場合があります。

```
Error: TypeError: speech.processExpression is not a function
```

この場合は `node_modules/mathjax_node/mj-single.js` の `GetSpeech` 関数にある `speech.processExpression(result.mml)` を `speech.toSpeech(result.mml)` に書き換えます。

24 Tips

24.1 コンテンツの幅を調整

スタイルシートで調整します。

```
.container {  
  max-width: 1200px;  
}
```

24.2 タグを無視

gitbook は `{%` と `%}` で囲んだ文字をタグと認識します。ですが、場合によっては無視したいこともあります。その場合は `raw` , `endraw` タグで囲みます。ただし、上手くいかないこともあります。

25 PDF

gitbook は Calibre というアプリケーションを使って電子文書形式を作成することができます。そのため、まずは Calibre を準備する必要があります。

- Calibre

Windows版のインストーラをダウンロードしてインストールします。ポータブル版でも構いません。ここでは "C:\Calibre" にインストールしたとします。

gitbook が Calibre を使用できるようにパスを通す必要があります。環境変数 PATH に C:\Calibre\Calibre を追加します。

これで各電子文書を作成することができます。PDFを作成する場合は次のコマンドを実行します。

```
gitbook pdf
```

上手く動作しない場合はPCを再起動してみましょう。環境変数が反映されていないかもしれません。

25.1 Anaconda用の設定

個人的にシステム環境変数やユーザー環境変数の PATH に追加することはあまり好きではありません。なるべく、PATH の設定は最小限にしたいと思っています。

今回は Anaconda を使っていて、gitbook という仮想環境を構築しています。そこで、gitbook の仮想環境のときに、Calibre のパスが通るようにします。

gitbook が有効の状態にします。

```
$ conda activate gitbook
```

仮想環境のフォルダを取得します。

```
$ (gitbook) set CONDA_PREFIX  
CONDA_PREFIX=C:\Miniconda3\envs\gitbook
```

C:\Miniconda3\envs\gitbook\etc\conda\activate.d というフォルダを作成し、そのフォルダ内に env_vars.bat というファイルを作成します。env_vars.bat の中身は次のようにします。

```
set PATH=C:\Calibre\Calibre;%PATH:C:\Calibre\Calibre;=%
```

このバッチファイルは activate したときに呼ばれるバッチファイルです。これで activate したときに PATH に Calibre が追加されます。

また、同様に C:\Miniconda3\envs\gitbook\etc\conda\deactivate.d というフォルダを作成し、env_vars.bat を作成すれば、deactivate されたときに呼ばれます。

25.2 Mathjax

Mathjax で SVG 形式で出力設定をしている場合、PDF出力時に以下のエラーが出ます。

```
Error: Error with command "svgexport"
```

そこで `svgexport` をインストールします。

```
npm install -y svgexport -g
```

25.3 Mermaid

どうやら pdf では図は出力されないようです。このような場合はあらかじめ画像で出力しておいて表示するといった対応が必要になります。

25.4 出力設定

`book.json` の `pdf` で設定できます。例えば、本文書では次のような設定になっています。

```
{
  "pdf": {
    "pageNumbers": true,
    "fontFamily": "VL Gothic",
    "fontSize": 10,
    "paperSize": "a4",
    "margin": {
      "right": 0,
      "left": 0,
      "top": 0,
      "bottom": 0
    },
    "headerTemplate": null,
    "footerTemplate": null
  }
}
```

PDF 出力時のスタイルシートは

```
{
  "styles": {
    "pdf": "styles/pdf.css"
  }
}
```

とすれば、`pdf.css` で編集できます。

26 参考

- [GitBook](#)
- [gitbook-cli](#)
- [gitbook-plugin-theme-api](#)
- [gitbook-plugin-theme-official](#)
- [gitbook-plugin-prism](#)
- [gitbook-plugin-sunlight-highlighter](#)
- [gitbook-plugin-copy-code-button](#)
- [gitbook-plugin-include-codeblock](#)
- [gitbook-plugin-hide-published-with](#)
- [gitbook-plugin-anchors](#)
- [gitbook-plugin-navigator](#)
- [gitbook-plugin-page-treeview](#)
- [gitbook-plugin-atoc](#)
- [gitbook-plugin-intopic-toc](#)
- [gitbook-plugin-back-to-top-button](#)
- [gitbook-plugin-numbered-headings-for-web-and-books](#)
- [gitbook-plugin-expand-active-chapter](#)
- [gitbook-plugin-collapsible-chapters](#)
- [gitbook-plugin-hints](#)
- [gitbook-plugin-footnote-string-to-number](#)
- [gitbook-plugin-advanced-emoji](#)
- [gitbook-plugin-todo](#)
- [gitbook-plugin-uml](#)
- [gitbook-plugin-mermaid-gb3](#)
- [gitbook-plugin-mathjax](#)
- [gitbook-plugin-katex](#)
- [gitbook-plugin-graph](#)
- [GitBook related configuration and optimization](#)