

Asciidoctorと Gradleでつくる 文書執筆環境



はじめに

本文書は Asciidoc とその Ruby による実装である Asciidoctor を用いて Asciidoc 文書を執筆する環境を構築する手順を示します。実行環境は Windows、Linux、macOS の各 OS に対応しています。

この文書の手順より以下のことができるようになります。

Asciidoc 形式で執筆した文書を HTML/PDF 形式に変換

Asciidoc 文書変換用 Gradle スクリプト

テキストエディターで変換結果をリアルタイムにプレビューしながら文書を編集

Visual Studio Code と AsciiDoc 拡張設定

Asciidoc は表現力の高い文書をテキストファイルベースで執筆できるテキストプロセッサーです。他の軽量テキストプロセッサーが持たない文書間のインクルードやソースコードの挿入などの機能も有し、かつ簡潔です。特に技術文書の執筆には大きな力を発揮することでしょう。



Asciidoc の表現力を示すひとつの例は、このような脚注表現です。

一般的にテキストプロセッサーを用いた執筆環境を整えるには多くの準備が必要ですが、本文書の手順は極力初期導入するプロダクトを少なく、簡単に快適な執筆環境を整えられるよう考えられました。

具体的には文書の変換に、実行を JVM 環境だけに依存する AsciidoctorJ と Gradle を活用し、執筆環境については Visual Studio Code を用いることでリアルタイムに文書をプレビューしながら、最後にコマンド一つで HTML/PDF 化できるように準備しました。またさらに、クラウド統合環境である Gitpod や GitHub の Codespaces もサポートし、ウェブブラウザだけでの執筆・変換も実現しています。

手間がかかることが多い PDF 出力に関しても、フォントセットや禁則処理設定をプロジェクトに持たせることにより、実行環境に関わらず同一出力が得られるように調整しました。

Asciidoc による文書の執筆は形式的で効率が良く、また更新差分が取りやすいため、文書履歴管理や共同執筆環境としても有効です。このメリットをさらに活用するため、本手順ではテキストダイアグラム記法による図形や数式のベクター画像挿入をサポートしています。

本文書がみなさんの執筆活動のお手伝いになれば幸いです。

ライセンス

本文書に含まれる筆者が作成したビルドスクリプト、サンプル文書.adoc、ダイアグラムなどのソースは全て**MIT License**です。ライセンスに従って自由にご利用ください。

リポジトリのソースやライセンスファイルは次から参照できます。

<https://github.com/h1romas4/asciidoc-gradle-template>

AsciidocとGradleでつくる文書執筆環境

– <https://github.com/h1romas4>



本文書のPDF版の表紙は@Fujixさんによって作成されました。

素敵なデザインをありがとうございました。

謝辞

本文書の実装であるビルドスクリプトやテーマでは次のプロダクトと技術資料が使われています。

プロダクト名の隣にライセンスを併記しました。商用利用等で制限のあるプロダクトはありませんが、それぞれライセンスを確認してください。

Font

- 源真ゴシック - SIL Open Font License 1.1 - <http://jikasei.me/font/genshin/>
- 源様明朝 - SIL Open Font License 1.1 - <https://github.com/ButTaiwan/genyo-font>
- Ricty Diminished - SIL Open Font License 1.1 - <https://github.com/edihbrandon/RictyDiminished>
- Morisawa BIZ UDGothic - SIL Open Font License 1.1 - <https://github.com/googlefonts/morisawa-biz-ud-gothic>
- Morisawa BIZ UDMincho - SIL Open Font License 1.1 - <https://github.com/googlefonts/morisawa-biz-ud-mincho>
- Open Iconic - MIT License - <https://github.com/iconic/open-iconic/>

Asciidoc

- Asciidoc - MIT License - <https://asciidoc.org/>
- Asciidocj - Apache License 2.0 - <https://github.com/asciidoc/asciidocj>
- Asciidoc.js - MIT License - <https://asciidoc.org/docs/asciidoc.js/>
- Asciidoc PDF - MIT License - <https://asciidoc.org/docs/asciidoc-pdf/>

- Asciidoctor Gradle Plugin Suite - Apache License 2.0 - <https://github.com/asciidoctor/asciidoctor-gradle-plugin>
- asciidoctor-pdf-linewrap-ja - MIT License - <https://github.com/fuka/asciidoctor-pdf-linewrap-ja>
- asciidoctor-nabetani - MIT License - <https://github.com/nabetani/asciidoctor-nabetani>

Diagram

- PlantUML - Multi(MIT/Apache/BSD/EPL/GPLv3/LGPL) - <https://github.com/plantuml/plantuml>
- ditaa - LGPL-3.0 license - <https://github.com/stathissideris/ditaa>

Build Tool

- SDKMAN - Apache License 2.0 - <https://sdkman.io/>
- Gradle - Apache License 2.0 - <https://gradle.org/>

Text Editor

- Visual Studio Code - Microsoft - <https://code.visualstudio.com/>
- asciidoctor-vscode - MIT License - <https://github.com/asciidoctor/asciidoctor-vscode>
- vscode-paste-image - MIT License - <https://github.com/mushanshitiancai/vscode-paste-image>
- vscode-plantuml - MIT License - <https://github.com/qjebbs/vscode-plantuml>

Guide

- asciidoctor-pdfでかっこいいPDFを作る - <https://qiita.com/kuboaki/items/67774c5ebd41467b83e2>

素晴らしい成果を公開されているみなさまに感謝します。

目次

はじめに	1
ライセンス	2
謝辞	2
1. Asciidoc 文書変換用スクリプトを使う準備	5
1.1. Java 実行環境の導入 (macOS / Linux)	6
1.2. Java 実行環境の導入 (Windows)	8
2. Asciidoc から HTML/PDF 文書を作成する	12
2.1. サンプル文書の変換を試す	12
2.2. 変換後の文書	15
2.3. テキストエディタで Asciidoc 文書を編集する	17
2.4. 文書のファイル構成	26
2.5. 執筆の開始	28
2.6. 執筆のイテレーション	28
2.7. 文書の Git 管理	30
2.8. クラウド環境による執筆	35
2.9. クラウド環境でのファイル保存と執筆の再開	39
2.10. GitHub Actions によるビルド	42
3. Asciidoc 記法	45
3.1. 文書中の相互参照	45
3.2. 外部リンク	46
3.3. 引用	46
3.4. 画像	47
3.5. ソースコードシンタックスハイライト	47
3.6. インラインコマンド・ファイル名	48
3.7. メニュー・ボタン	48
3.8. ファイルインクルード	48
3.9. 表形式	49
3.10. リスト	50
3.11. 脚注	52
3.12. 改ページ・水平線	53
3.13. コラム表現	53
3.14. コメント行	53
4. ダイアグラム記法	54
4.1. シーケンス図	55
4.2. フォルダ・ファイルツリー	57
4.3. 数式	58
4.4. ブロック図	59
4.5. メモリーマップ	61
4.6. ネットワーク構成図	62
4.7. システム構成図	64
4.8. タイミングチャート	66
4.9. マインドマップ	68
奥付	69

1. Asciidoc 文書変換用スクリプトを使う準備

本手順で用いる Asciidoc 文書変換用スクリプトはビルドツールである Gradle を活用しており、実行するためには Java 実行環境が必要です。

お使いのコンピューターのコマンドライン環境（macOS/Linux ではターミナル、Windows では cmd.exe か powershell.exe）で `java -version` コマンドを入力し、Java 11 以上のバージョンが表示されるようであれば既に準備は整っています。

macOS/Linux の場合

```
$ java -version
openjdk 11.0.20.1 2023-08-24
OpenJDK Runtime Environment Temurin-11.0.20.1+1 (build 11.0.20.1+1)
OpenJDK 64-Bit Server VM Temurin-11.0.20.1+1 (build 11.0.20.1+1, mixed mode)
```

Windows の場合

```
C:\> java -version
openjdk version "11.0.20.1" 2023-08-24
OpenJDK Runtime Environment Temurin-11.0.20.1+1 (build 11.0.20.1+1)
OpenJDK 64-Bit Server VM Temurin-11.0.20.1+1 (build 11.0.20.1+1, mixed mode)
```



本文書では Java 11 を用いて解説します。また、ビルド時に僅かに内部処理のワーニングが出力されますが Java 17 でも期待通り動作することを確認しています。Java 21 に関しては周辺ツールの対応待ちのステータスです。

Graphviz の導入

macOS 及び Linux の場合で、後述する PlantUML 以外のダイアグラム記法を使う場合は Graphviz の導入が必要です。この文書のサンプルソースでも使っている部分が一部ありますので、ワーニングなしに本文書の完全なファイルを生成したい場合は次のようにインストールしてください。Windows の場合、この操作は不要です。

macOS

```
brew install graphviz
```

Ubuntu

```
sudo apt install graphviz
```

1.1. Java 実行環境の導入(macOS / Linux)

もし macOS / Linux 環境に Java 実行環境がなければ SDKMAN を利用することで、ターミナルから簡単に導入できます。

<https://sdkman.io/>

SDKMAN! is a tool for managing parallel versions of multiple Software Development Kits on most Unix based systems.

— SDKMAN

手順. SDKMAN を用いた Java の導入

```
$ curl -s "https://get.sdkman.io" | bash ①
$ source "$HOME/.sdkman/bin/sdkman-init.sh" ②
$ sdk list java ③
=====
Available Java Versions for Linux 64bit
=====
Vendor      | Use | Version      | Dist   | Status    | Identifier
-----
Temurin     |     | 20.0.2        | tem    |           | 20.0.2-tem
|     | 20.0.1        | tem    |           | 20.0.1-tem
|     | 17.0.8        | tem    | installed  | 17.0.8-tem
|     | 17.0.8.1      | tem    |           | 17.0.8.1-tem
|     | 17.0.7        | tem    | installed  | 17.0.7-tem
|     | 17.0.4        | tem    | local only | 17.0.4-tem
|     | 11.0.20       | tem    |           | 11.0.20-tem
| >>> | 11.0.20.1     | tem    | installed  | 11.0.20.1-tem
|     | 11.0.19       | tem    | installed  | 11.0.19-tem
$ sdk install java 11.0.20.1-tem ④
```

- ① SDKMAN を導入します。
- ② SDKMAN を環境に設定します。
- ③ 導入できる Java のバージョンを一覧します。
- ④ Java 11 系の最新バージョンを指定して Java を導入します。

また、Gradle は JAVA_HOME 環境変数に実行環境の Java のパスが設定されていることを期待していますので、利用しているシェル環境に合わせ `.bashrc` や `.zshrc` 等で次のように JAVA_HOME を設定します。

手順. JAVA_HOME の設定

```
$ vi ~/.bashrc ①  
export JAVA_HOME=~/.sdkman/candidates/java/current ②  
$ source ~/.bashrc ③
```

- ① エディタで `.bashrc` や `.zshrc` を開きます。
- ② 本ラインをファイルの最下部に追加し保存終了します。
- ③ 設定を適用します。`source` コマンドを使わず、ターミナルを再起動する操作でも反映されます。

以上で準備完了です。

SDKMANについて

SDKMAN は主に Java エコシステムの開発環境をコマンドラインから簡単に導入・設定するためにつくられた管理ソフトウェアです。

たとえば次のように簡単に各種 Java のバージョンを導入し切り替えできます。

手順. SDKMAN による Java のバージョン切り替え

```
$ sdk install java 11.0.20.1-tem ①  
$ sdk default java 11.0.20.1-tem ②  
$ sdk default java 17.0.4.1-tem ③
```

- ① Java 11 を導入
- ② Java 11 をデフォルトに設定
- ③ Java 17 をデフォルトに戻す

1.2. Java 実行環境の導入(Windows)

もし Windows 環境に Java 実行環境がなければ AdoptOpenJDK プロジェクトが提供する OpenJDK のバイナリを導入すると良いでしょう。

<https://adoptopenjdk.net>

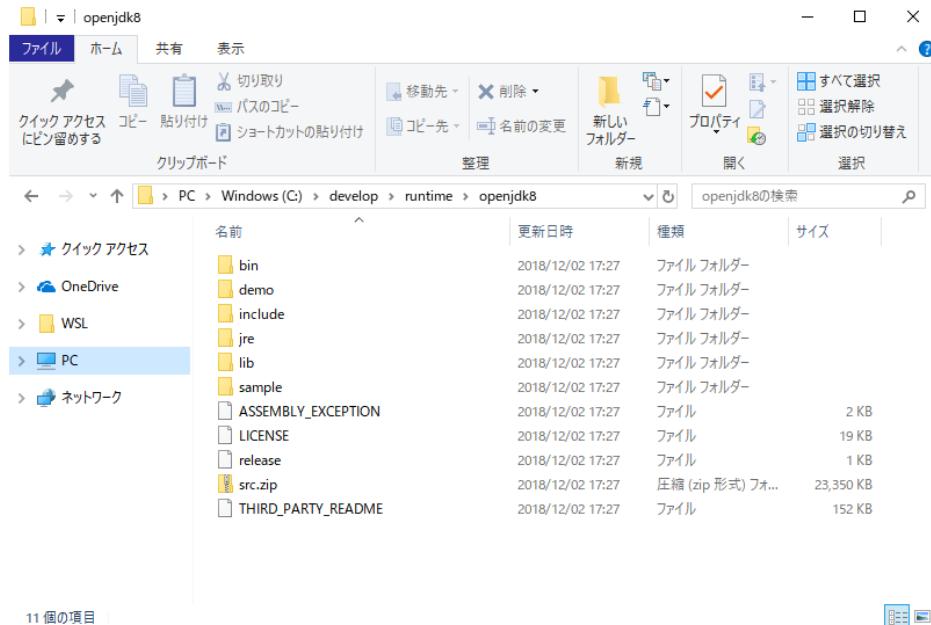
Java™ is the world's leading programming language and platform. The code for Java is open source and available at OpenJDK™. AdoptOpenJDK provides prebuilt OpenJDK binaries from a fully open source set of build scripts and infrastructure.

— AdoptOpenJDK

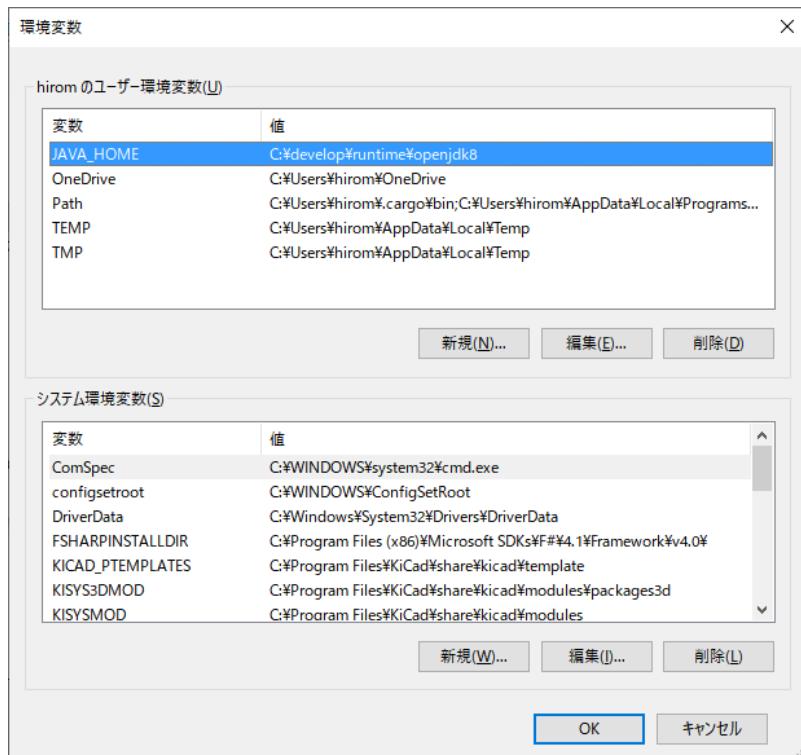
<https://adoptopenjdk.net> サイトにブラウザでアクセスし、OpenJDK 11 (LTS) - HotSpot を選択した後、zip ファイルをダウンロードしてください。

The screenshot shows the AdoptOpenJDK website's "Prebuilt OpenJDK Binaries" page. At the top, there is a dark header with the AdoptOpenJDK logo and a Twitter icon. Below the header, the title "Prebuilt OpenJDK Binaries" is centered. A brief description follows, mentioning Java's status as the world's leading programming language and platform, its open source nature, and availability at OpenJDK™. It also links to Docker images on Docker Hub and the Archive. A "Downloads" section is present, with two main steps: "1. Choose a Version" and "2. Choose a JVM". Under "Choose a Version", "OpenJDK 8 (LTS)" is selected (indicated by a blue radio button). Under "Choose a JVM", "HotSpot" is selected. A large blue button at the bottom left of the "Downloads" section says "Latest release" with a refresh icon, and below it, the text "jdk8u192-b12".

zip ファイルを任意の場所に展開します。ここでは C:\develop\runtime\openjdk11 に展開したとします。



Gradle は JAVA_HOME 環境変数に実行環境の Java のパスが設定されていることを期待していますので、エクスプローラー › PC (右クリック) › プロパティー › 詳細設定 › 環境設定 からユーザー環境変数に JAVA_HOME を追加し、先ほど .zip を展開したパス (C:\develop\runtime\openjdk11) を設定します。



Gradle は JAVA_HOME 環境変数を元に Java の実行環境を探すため、java コマンドを使うための PATH 環境変数は設定しなくてもかまいません。

以上で準備完了です。

1.2.1. Windows 環境の WSL2 上の Ubuntu を使う

Windows 環境では WSL2 Linux 仮想環境上の Ubuntu 22.04 を使うことでも環境構築が可能です。メインの執筆環境が macOS や Ubuntu の場合に、Windows 上でも WSL2 Ubuntu を使うと各種操作が統一できて便利かもしれません。

WSL2 に Ubuntu 22.04 LTS が導入済みであることを前提に、環境構築手順は次のようになります。

WSL2 Ubuntu 22.04 LTS の初期設定

```
$ sudo apt update
$ sudo apt upgrade
$ sudo apt install language-pack-ja graphviz fontconfig fonts-noto* language-selector-common ①
$ sudo update-locale LANG=ja_JP.UTF8 ②
$ echo 'export LANG=ja_JP.UTF-8' >> ~/.bashrc ③
$ fc-cache -f ④
```

- ① 日本語の言語設定とデフォルトフォント関連の導入。
- ② 日本語ロケールを追加。
- ③ 起動シェル環境を ja_JP.UTF-8 に設定。
- ④ 念のためフォントキャッシュを更新。



これらの日本語ロケール設定は、後述のダイアログ表記を使う場合に出力される図表内のフォントが正しく配置されるように行っています。

次に、通常の Ubuntu の手順と同様に sdkman で Java 環境の導入を行います。

WSL2 Ubuntu 22.04 LTS への Java の導入

```
$ curl -s "https://get.sdkman.io" | bash ①
$ source "$HOME/.sdkman/bin/sdkman-init.sh" ②
$ sdk install java 11.0.20.1-tem ③
$ echo 'export JAVA_HOME=~/.sdkman/candidates/java/current' >> ~/.bashrc ④
```

- ① SDKMAN を導入。
- ② SDKMAN を環境に設定。
- ③ Java 11 系を指定して Java を導入。
- ④ JAVA_HOME を設定。

この操作後 LANG と JAVA_HOME 環境変数設定を反映させるため一度シェルを再起動してください。

以上で準備完了です。

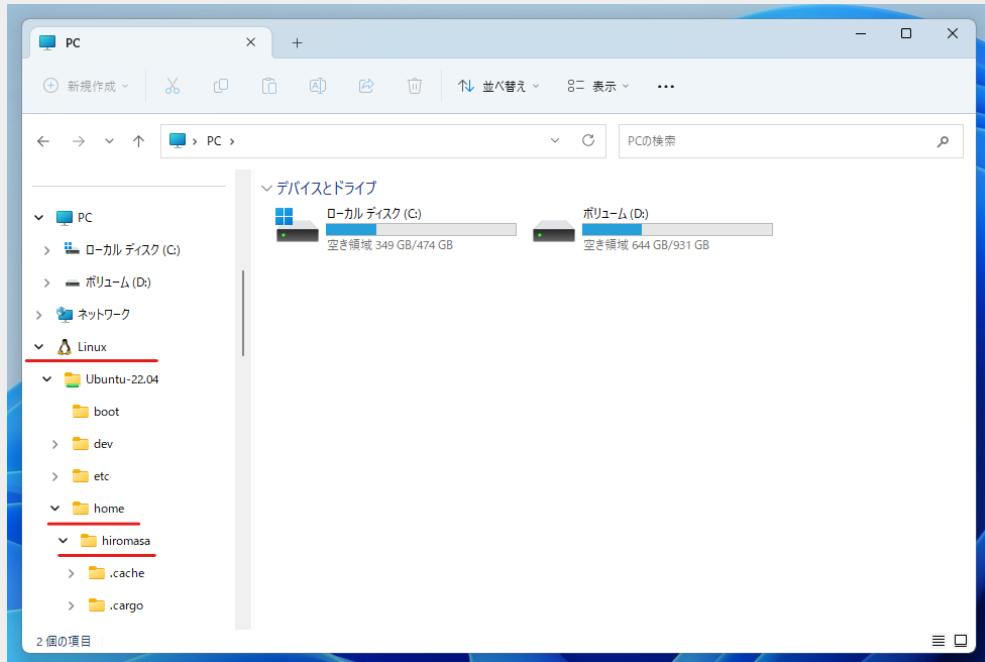


次項以降 WSL2 Ubuntu 環境を使う場合の手順は、特別な記載がある場合を除き Linux 項を参照して操作してください。

WSL2 Ubuntu 環境の作業ディレクトリ

WSL2 Ubuntu 上で文書の編集などを行う作業ディレクトリは、Ubuntu 側のホームディレクトリ（`/home/[ユーザ名]/`）配下が良いでしょう。`/mnt/c/` から見える Windows 側のファイルシステムを作業フォルダとして使うこともできますが、I/O 処理速度がシミュレーションにより遅くなるため文書の変換処理のボトルネックになる場合があります。

この構成をとった場合、Windows 側のエクスプローラから Ubuntu のファイルシステムにアクセスする機能が便利です。エクスプローラの操作で、WSL2 上で作成した文書を Windows 側で簡単に取得できます。



画面. Windows エクスプローラからみた WSL2 ファイルシステム

2. Asciidoc から HTML/PDF 文書を作成する

2.1. サンプル文書の変換を試す

本項では準備した環境を元に、Asciidoc 文書を HTML/PDF に変換する手順を示します。

変換に使うスクリプトは GitHub のリポジトリに公開されており、HTML/PDF 変換に使うファイル一式と、文書のサンプルとして "この文書" の Asciidoc ファイルが配置されています。

最初にリポジトリからこれらのファイルの取得を行い、次に文書の変換のために Gradle タスクを実行します。各 OS ごとの代表的な操作は次のようになります。

2.1.1. サンプル文書の変換(macOS / Linux)

ターミナルを起動し curl コマンドでファイルの取得を行い、ファイルの展開と Gradle タスクの実行を行います。

手順. PDF 変換ビルドスクリプトの取得と実行

```
$ curl -L -O https://github.com/h1romas4/asciidoc-gradle-template/archive/main.zip ❶
$ unzip main.zip ❷
$ cd asciidoctor-gradle-template-main ❸
$ ./gradlew docs ❹
BUILD SUCCESSFUL in 19s ❺
2 actionable tasks: 2 executed
```

- ❶ リポジトリのファイルをダウンロードします。
- ❷ ダウンロードした .zip ファイルを展開します。
- ❸ カレントディレクトリを展開したフォルダの中に移します。フォルダ名は任意のものに変更可能です。本手順ではプロジェクトフォルダと呼称します。
- ❹ Gradle のビルドを実行します。初回実行時はビルドに必要なファイルをダウンロードするため少し時間がかかります。次回以降は30秒程度で完了します。
- ❺ BUILD SUCCESSFUL が出力されればビルド成功です。

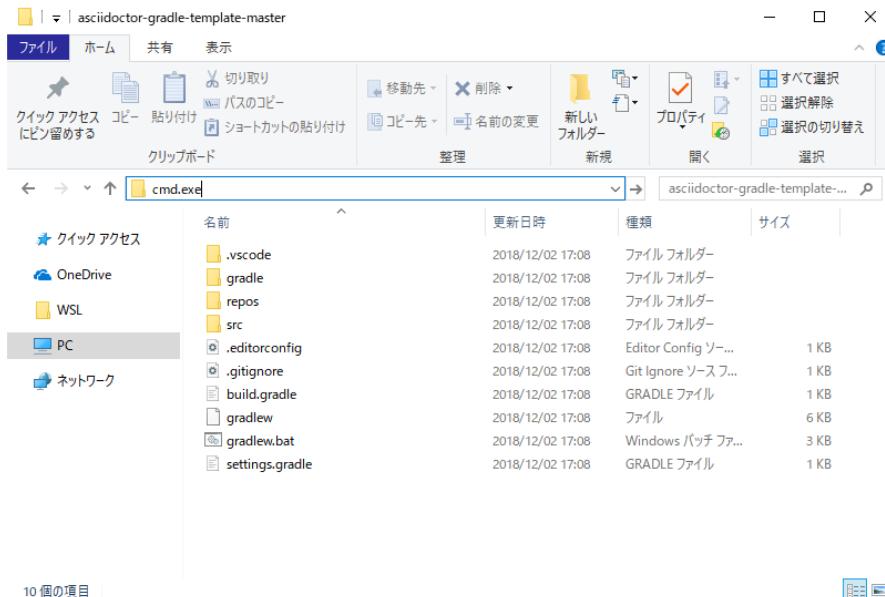
2.1.2. サンプル文書の変換(Windows)

ウェブブラウザで URL にアクセスしてファイルの取得を行い、エクスプローラでファイルを展開し、最後にコマンドプロンプトから Gradle タスクを実行します。



Windows の場合 .zip ファイルの展開先はマルチバイト文字を含まないパスにしてください。JRuby の制約により変換処理がエラーとなります。また、プロジェクト内部で使うフォルダやファイル名のマルチバイト名も同様です。

1. ブラウザを使って <https://github.com/h1romas4/asciidoc-gradle-template/archive/main.zip> にアクセスしリポジトリのファイルを取得します。
2. ダウンロードした .zip ファイルを右クリックし展開します。フォルダ名は任意のものに変更可能です。本手順ではプロジェクトフォルダと呼称します。
3. 展開したフォルダ内をエクスプローラーで表示した上で、アドレスバーに cmd.exe と入力し、このフォルダをカレントディレクトリとしてコマンドプロンプトを起動します。



4. `\gradlew.bat docs` と入力し Gradle ビルドを実行します。初回実行時はビルドに必要なファイルをダウンロードするため少し時間がかかります。次回以降は30秒程度で完了します。
5. BUILD SUCCESSFUL が出力されればビルド成功です。

プロキシサーバーの設定

もしお使いのコンピューターがプロキシサーバー経由のインターネットアクセスを行う場合は、次のコマンドを `./gradlew docs` をする前に入力してください。インターネットを使ったライブラリの取得が正しく行われるようになります。ホスト名 (`example.com`) と ポート番号 (8080) 部分はそれぞれの環境に合わせてください。

プロキシサーバー設定(macOS / Linux)

```
export JAVA_OPTS=-DproxyHost=example.com -DproxyPort=8080
```

プロキシサーバー設定(Windows)

```
set JAVA_OPTS=-DproxyHost=example.com -DproxyPort=8080
```

2.2. 変換後の文書

./gradlew docs のビルド操作により Asciidoc から変換された文書は、docs フォルダに HTML版(index.html)と PDF版 (index.pdf) として格納されます。



画像.HTML 文書



画像.PDF 文書

成果物の出力先となる `docs` フォルダの構成は次のとおりです。

docs	
Chapter00	章ごとの HTML 版用のリソース
Chapter01	
images	HTML 版用の画像格納フォルダ
AdoptOpenJDK.png	HTML 版用の PNG 画像リソース
windows-01.png	
windows-02.png	
Chapter02	
images	
html.png	
pdf.png	
..snip..	
Chapter03	
images	
diag-salt-sample1.svg	HTML 版向けに自動生成されたベクター画像
diag-salt-sample2.svg	後述のダイアグラム記法による
..snip..	
index.html	HTML 版(CSSを内包・画像はファイルに依存)
index.pdf	PDF 版(全ての画像・フォントリソースを内包)

成果物文書のソースとなるのは `src/docs/asciidoc` に配置された Asciidoc 文書と画像ファイル群です。

```
graph TD; src[src] --> docs[docs]; docs --> asciidoc[asciidoc]; asciidoc --> Chapter00[Chapter00]; Chapter00 --> index00[index.adoc]; asciidoc --> Chapter01[Chapter01]; Chapter01 --> index01[index.adoc]; asciidoc --> images[images]; images --> AdoptOpenJDK.png[AdoptOpenJDK.png]; images --> windows01[windows-01.png]; images --> windows02[windows-02.png];
```

章ごとの AsciiDoc 文書

文書で使う画像ファイル

`docs` 出力フォルダの構成としては、章 (`Chapter*/images`) フォルダには HTML 版からリンクされるリソースが `src/docs/asciidoc` からコピーされ配置されます。また、後述するダイアグラム記法を使った図表や数式も、ビルドのタイミングでベクター画像が生成され同フォルダに出力されます。

PDF版のファイルについては index.pdf 単体で完結しており、フォントや画像リソースなど全てが PDF ファイル内に格納されます。

以上から成果物の配布は次のようにになります。

HTML 文書	docs フォルダの index.pdf を除く全てのファイルを配布。
PDF 文書	docs/index.pdf のみを配布。

なおいずれの場合も `index.*` のファイル名は任意の名称に変更可能です。



`docs` フォルダは成果物の出力専用フォルダとなっており、ビルト時にいったん全てのファイルが削除されますので、自らが作成したファイルは配置しないように注意してください。執筆で作成するファイルは必ず `src/docs/asciidoc` 配下に配置します。

2.3. テキストエディタで Asciidoc 文書を編集する

2.3.1. Visual Studio Code の導入(macOS / Linux / Windows)

Visual Studio Code（以下 VS Code）は、Microsoft が提供する高性能なフリーソフトウェアのテキストエディタです。

本手順のプロジェクトフォルダには VS Code 用の設定ファイル (`.vscode` 配下) が配置されており、自動的に Asciidoc 文書向けの設定がされるように構成されています。

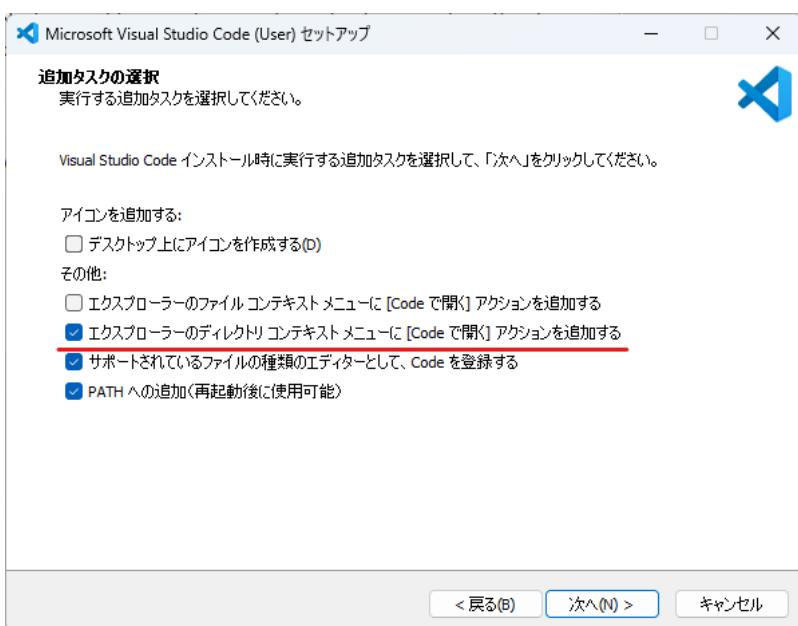
次のサイトからお使いの OS に合わせた VS Code を導入してください。

<https://code.visualstudio.com/>

Free. Built on open source. Runs everywhere.

— Visual Studio Code - Code Editing. Redefined

インストールプロセスではオプションを次のように指定し、コンテキストメニューに VS Code を追加すると今後の操作が便利になります。



VS Code ではプロジェクトファイルとして「フォルダを開く」操作をよく活用します。このことから、エクスプローラなどのシェルのフォルダ右クリックのコンテキストメニューに VS Code を追加すると素早い操作がで

きるようになります。

以上で、VS Code の準備は完了です。

2.3.2. Visual Studio Code の導入(WSL2 Ubuntu 環境)

WSL2 環境上の Ubuntu で VS Code を使う場合は「Windows 版 VS Code」を導入後さらに次の WSL 拡張をインストールします。

<https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.remote-wsl>

The WSL extension lets you use VS Code on Windows to build Linux applications that run on the Windows Subsystem for Linux (WSL). You get all the productivity of Windows while developing with Linux-based tools, runtimes, and utilities.

— WSL - Visual Studio Marketplace

拡張導入後、一度 VS Code を終了してください。

次に、WSL2 Ubuntu のターミナル画面で `code` コマンドを入力することで WSL2 Ubuntu 上で VS Code サーバの初期セットアップが動作しその後 VS Code 画面が立ち上がります。

WSL2 Ubuntu 22.04 への VS Code サーバの導入

```
$ code ①
```

① WSL Ubuntu 環境上から Windows 側の VS Code を起動する。

以上で、VS Code の準備は完了です。

ここから先の手順で VS Code を起動する場合は同様に WSL2 Ubuntu のターミナル画面から `code` コマンドで起動してください。

リスト 1. WSL2 Ubuntu 22.04 からの VS Code 起動

```
$ cd asciidoctor-gradle-template ①
$ code . ②
```

① 編集したいプロジェクトにカレントディレクトリを移します。

② `.` とカレントディレクトリを指定して VS Code を起動します。

なお、VS Code WSL2 拡張の動作の詳細について知りたい場合は以下のドキュメントを参照してください。

<https://code.visualstudio.com/docs/remote/wsl>

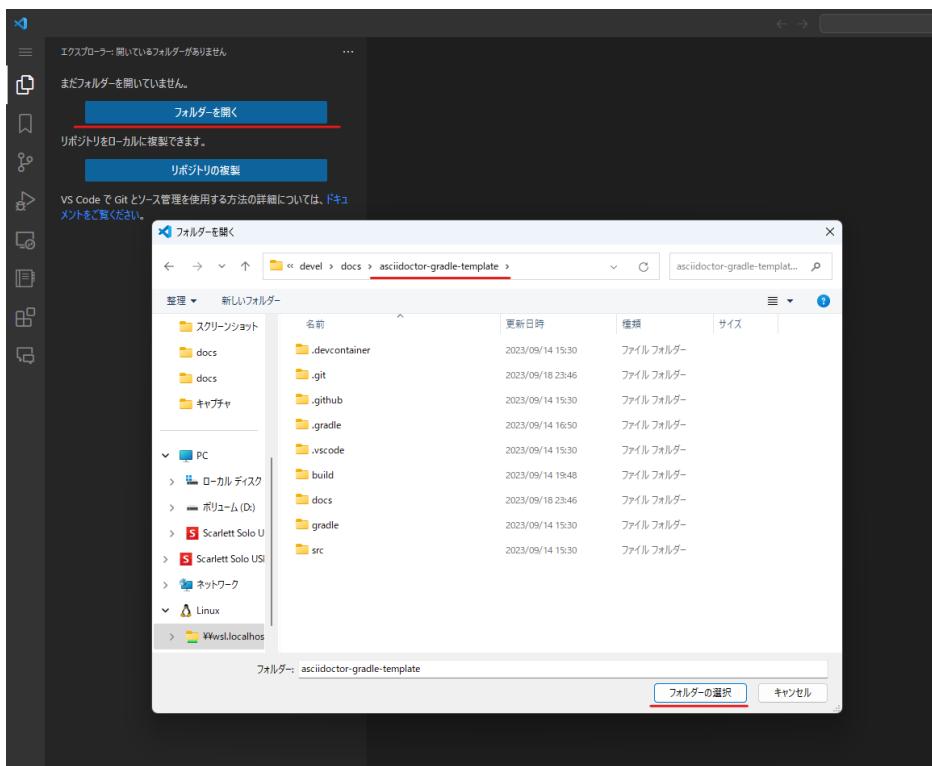
The Visual Studio Code WSL extension lets you use the Windows Subsystem for Linux (WSL) as your full-time development environment right from VS Code. You can develop in a Linux-based environment, use Linux-specific toolchains and utilities, and run and debug your Linux-based applications all from the comfort of Windows.

— Developing in the Windows Subsystem for Linux with Visual Studio Code

2.3.3. VS Code による Asciidoc 文書のリアルタイムプレビュー

本手順のプロジェクトフォルダに配置された Asciidoc 文書は、VS Code を利用してリアルタイムに変換結果をプレビューしながら編集できるように設定されています。

最初のステップとして、プロジェクトフォルダを VS Code を開きます。VS Code でプロジェクトを開く場合は、次のように「フォルダで開く」などの操作でプロジェクトフォルダがルートになる形で行うことに注意してください。



プロジェクトフォルダのオープンは、エクスプローラなどのシェルからフォルダを右クリックして表示されるコンテキストメニューの [Code で開く] 操作や、ターミナル画面からカレントディレクトリを `cd` コマンドで移動し `code .` と入力することでも可能です。

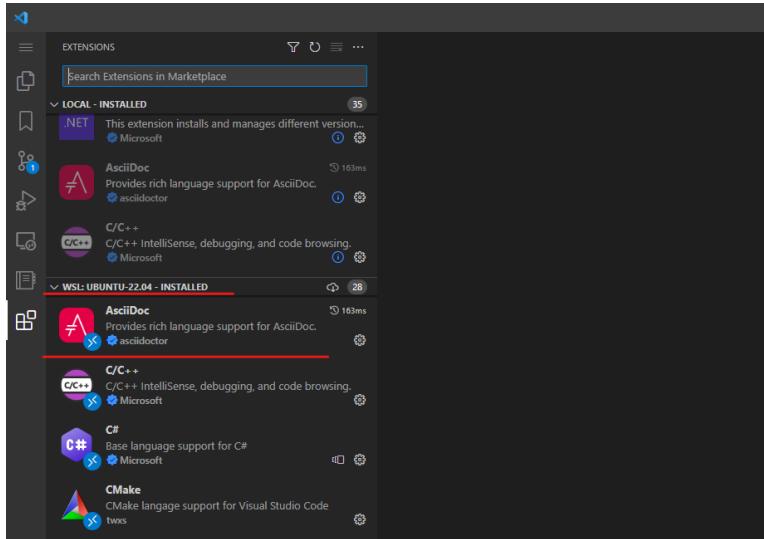
次に VS Code から推奨拡張機能に関する問い合わせが表示されますので、確認の上、次のような操作で導入してください。

- このワークスペースには拡張機能の推奨事項があります。>すべてインストール ボタンをクリックします。

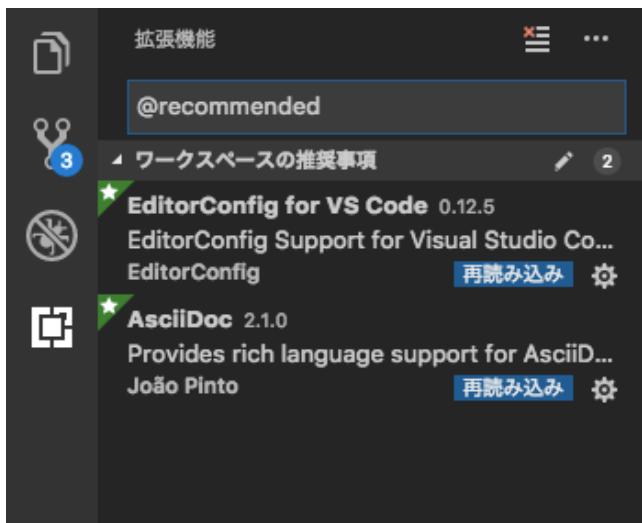
2. Asciidoc から HTML/PDF 文書を作成する



WSL2 Ubuntu 環境を使っている場合は合わせて WSL2 側にも拡張の導入を行います。



2. [再読み込み] ボタンをクリックします。



導入推奨拡張は `.vscode/extension.json` ファイルで設定されています。文書に応じて設定し、執筆メンバーの環境を揃えられます。

以上で Asciidoc 文書編集のための準備が完了しました。

VS Code 左パネルのエクスプローラーから、Asciidoc 文書（src/docs/asciidoc/index.adocなど）を開き、文書を開いたエディタ部右上に配置された [Open Preview to the Side] アイコンをクリックすると、画面右側に Asciidoc 文書の変換がリアルタイムに確認できるプレビューが表示されます。



プレビュー表示操作はキーボードショートカット [Ctrl + k, v] でも可能です。また、画面右側ではなく現在のエディタグループに表示する場合は [Ctrl + Shift + v] を押下します。

The screenshot shows the VS Code interface with two tabs open: 'index.adoc .../Chapter03' and 'index.adoc .../Chapter01'. The main editor area contains Asciidoc code, and the right-hand preview pane displays the resulting HTML content. A callout box highlights the terminal output of a build command:

```
$ curl -L -O https://github.com/h1romas4/asciidoctor-gradle-template/archive/master.zip # <1>
$ unzip master.zip # <2>
$ cd asciidoctor-gradle-template-master # <3>
$ ./gradlew docs # <4>
BUILD SUCCESSFUL in 19s <5>
2 actionable tasks: 2 executed
```

Below the terminal output, numbered steps explain the process:

- ① リポジトリのファイルをダウンロードします。
- ② ダウンロードした .zip ファイルを展開します。
- ③ カレントディレクトリを展開したフォルダの中に移します。
- ④ Gradle のビルドを実行します。初回実行時はビルドに必要なファイルをダウンロードするため少し時間がかかります。次回は数秒で完了します。
- ⑤ BUILD SUCCESSFUL が表示されればビルド成功です。

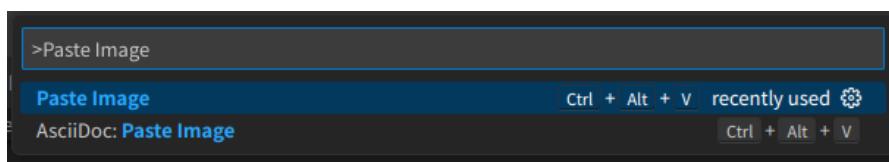
.adoc ファイルには本文書の内容がそのまま Asciidoc 文書形式で書かれているのが分かります。いくつか開いて試し入力などをしてみると、Asciidoc 文書のリアルタイムプレビューを含め執筆の雰囲気が掴めるはずです。

2.3.4. クリップボードからの画像挿入

本手順で VS Code の推奨拡張として導入される [Paste Image](#) (mushan.vscode-paste-image) は、クリップボード上にある画像をファイルとして指定の位置に格納した上で、Asciidoc 文書にリンクを挿入する便利な拡張です。

特にコンピュータの操作手順文書をつくる場合に活用すると強力です。次のような簡単な操作でクリップボードの画像を Asciidoc 文書に挿入できます。サンプル文書内で操作を試してみると良いでしょう。

1. スクリーンキャプチャなどの機能でクリップボードに画像を保持する。
2. VS Code で Asciidoc 文書を開き、VS Code > F1 キー押下 > Paste Image と入力 > エンターキーで選択する。



Asciidoc 文書には画像リンクが挿入されるとともに、開いている Asciidoc 文書から相対で `images/` として見えるパスに画像ファイルが格納され、文書への画像挿入がひとつの操作で完了します。

```
image::2023-06-30-12-11-56.png[]
```

画像ファイル名の命名規約や出力先のフォルダ設定は `.vscode/setting.json` で行うことができます。

```
.vscode/setting.json
```

```
{  
    "pasteImage.path": "${currentFileDir}/images",  
    "pasteImage.basePath": "${currentFileDir}/images",  
    "pasteImage.defaultName": "Y-MM-DD-HH-mm-ss"  
}
```

Asciidoc 拡張にも同様の機能（VS Code ➤ F1 キー押下 ➤ AsciiDoc: Paste Image）がありますが、現在のところ Paste Image 拡張のほうがファイル名などの設定が柔軟であるため、本手順では Paste Image を紹介しています。

AsciiDoc 版の Paste Image には現在の開いている Asciidoc の `:imagesdir:` 属性を見て設定無しでファイル配置場所を決定してくれるなど優れた機能もありますので、それぞれの機能を確認し必要に応じて使い分けると良いでしょう。

クリップボード連携のシステムインターフェース

VS Code Paste Image 拡張のクリップボード連携のシステムインターフェースは次のシェルスクリプトで実装されています。

```
$ ls -laF ~/.vscode/extensions/mushan.vscode-paste-image-1.0.4/res
..snip..
-rw-r--r-- 1 hiromasa hiromasa 492 6月 26 2021 linux.sh
-rw-r--r-- 1 hiromasa hiromasa 1540 6月 26 2021 mac.applescript
-rw-r--r-- 1 hiromasa hiromasa 680 6月 26 2021 pc.ps1
..snip..
```

Asciidoc 拡張も ~/.vscode/extensions/asciidoc.asciidoc-vscode-2.9.8/res 配下に同様のファイルがあります。これらの拡張のクリップボード連携は各 OS のクリップボード操作のコマンドを呼び出して動作しています。動作をご理解の上活用してください。

また、WSL2 Ubuntu と VS Code の WSL Remote 拡張を使った環境では、Paste Image は Windows - Ubuntu 間のクリップボードがそのままでは跨げないためうまく動作しません。

これを解決するためのワークアラウンドですが、次のように WSL2 側の Paste Image 拡張のファイルを書き換えると画像のクリップボード連携が動作するようになります。

```
~/.vscode-server/extensions/mushan.vscode-paste-image-1.0.4/res/linux.sh
```

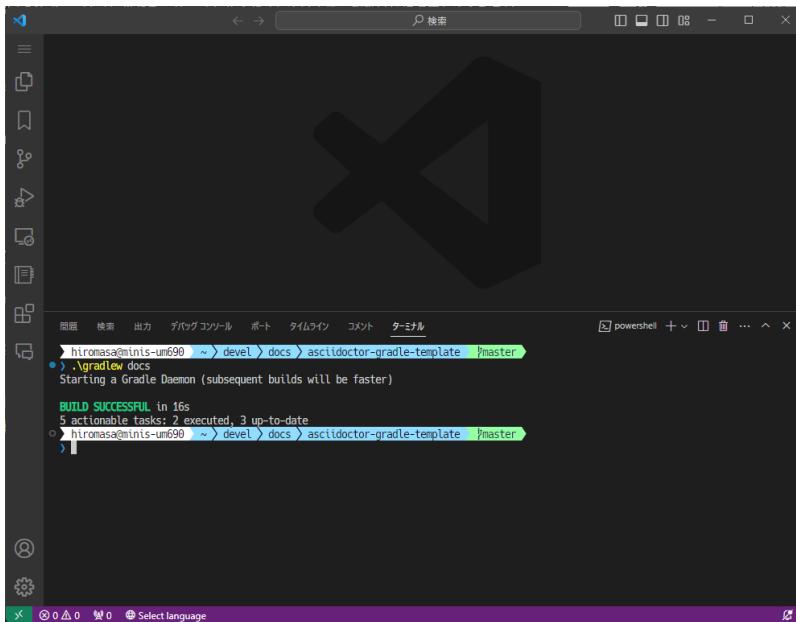
```
#!/bin/sh
# change current directory
cd `dirname $0`
# set dist path
DISTPATH=//wsl.localhost/Ubuntu-22.04 ①
# call Windows PowerShell
powershell.exe -ExecutionPolicy Bypass -File pc.ps1 ${DISTPATH}$1
```

① 利用しているディストリビューションのパスを設定する。

なお、これは暫定的な方法のため拡張にアップデートがあると修正したファイルが戻って機能しなくなります。また -ExecutionPolicy Bypass 指定より PowerShell のセキュリティをバイパスしています。取扱いには注意してください。

2.3.5. 統合ターミナルの活用

VS Code には統合ターミナルと呼ばれる VS Code 画面内で使えるターミナルエミュレータが備わっており、ここで文書変換コマンドを実行できます。



VS Code 統合ターミナルからの文書変換

統合ターミナルは **表示 > ターミナル** と操作して開きます。

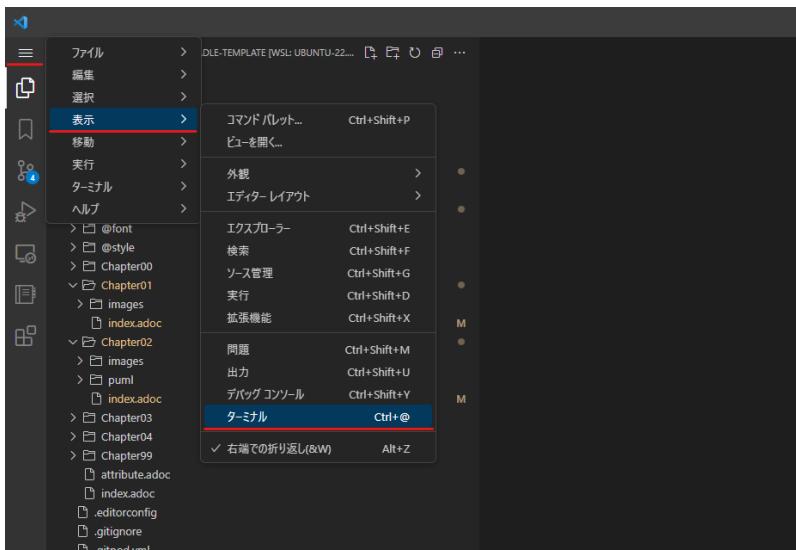


図 1. VS Code 統合ターミナルの起動

統合ターミナルを使うと VS Code を離れることなく文書の変換ができますので活用してみてください。

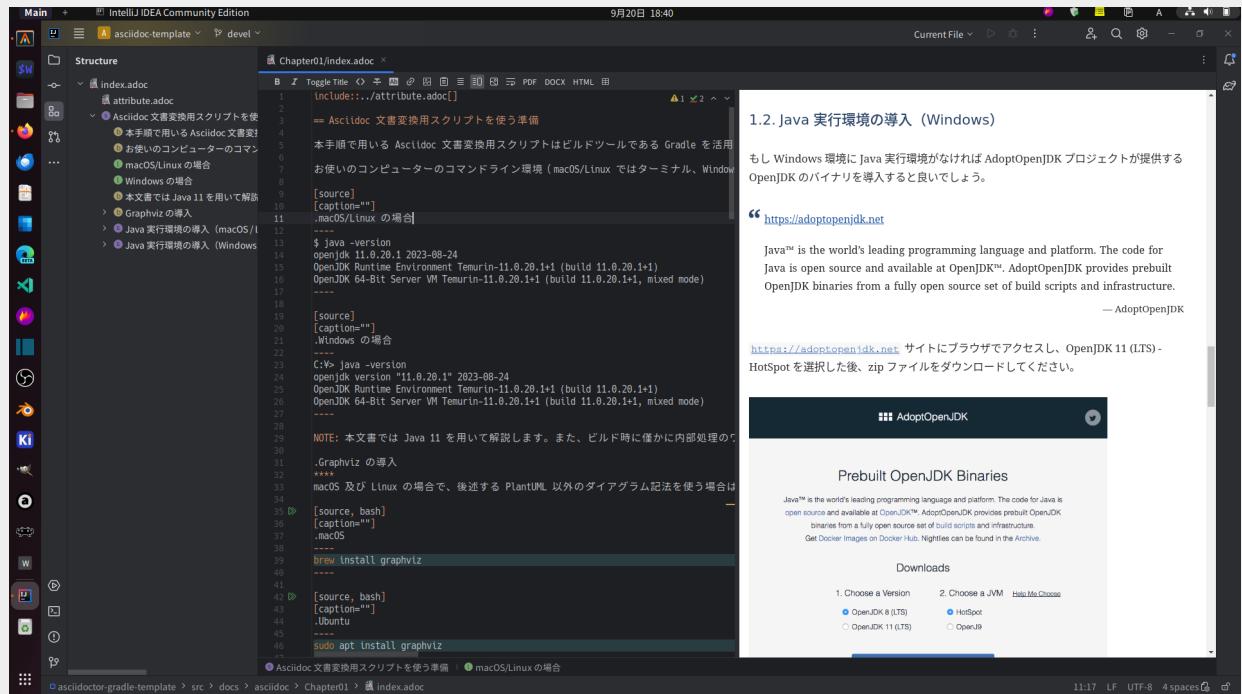
統合ターミナルはキーボードショートカット **[Ctrl + @]** でも起動可能です。



また、統合ターミナル内では **[↑]** を押下することで前回入力した履歴が呼び出せます。この操作は文書変換コマンドを再実行する際に便利です。

その他の Asciidoc 文書編集が可能な統合開発環境

本手順では VS Code を Asciidoc 文書編集用のエディタとして活用していますが、Eclipse や IntelliJ IDEA といった統合開発環境も Asciidoc を拡張機能でサポートしています。

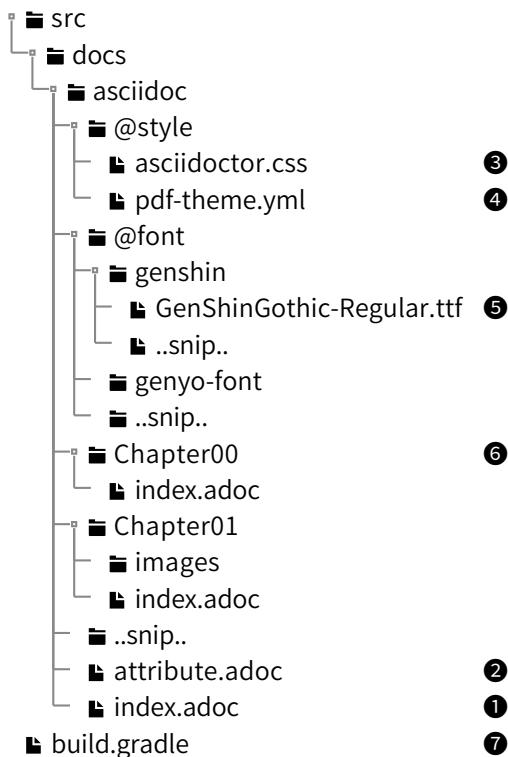


画像. IntelliJ IDEA の Asciidoc Plugin で文書編集している例

好きな統合開発環境があればプロジェクトフォルダにそれぞれの環境定義を入れ、自分に合った執筆環境を構築してみてください。

2.4. 文書のファイル構成

以下に執筆で編集する Asciidoc 文書フォルダ (src/docs/asciidoc 配下) のファイル構成を示します。



```

src/docs/asciidoc/index.adoc ①
src/docs/asciidoc/attribute.adoc ②
src/docs/asciidoc/@style/asciidoctor.css ③
src/docs/asciidoc/@style/pdf-theme.yml ④
src/docs/asciidoc/@font/**/*.ttf ⑤
src/docs/asciidoc/Chapter{number}/index.adoc ⑥
build.gradle ⑦

```

- ① 文書を作成する起点となる Asciidoc 文書です。表紙となる文書のタイトルなどを記載し、その後から章ごとの Asciidoc 文書を include して構成していきます。
- ② 文書設定をするためのファイルです。各文書から include します。
- ③ HTML 出力とプレビュー用のスタイルシートです。文書に合わせて修正できます。
- ④ PDF 文書に変換する際に使われるスタイル定義です。文書に合わせて修正できます。
- ⑤ PDF 文書に埋め込みされるフォントファイルを配置します。pdf-theme.yml からファイル名で参照されています。TrueType フォント .ttf が指定できます。
- ⑥ src/docs/asciidoc/index.adoc から参照される子文書です。build.gradle による画像パス解決のためフォルダ名は Chapter{number} とします。
- ⑦ 文書を変換する Gradle ビルドスクリプトです。Asciidoc 文書や画像パスの設定があります。

本構成を元に執筆したい文書に合わせカスタマイズしていきます。文書ファイルの編集や閲覧は VS Code のリアルタイムプレビューで確認しながら行うと分かりやすいでしょう。

2.4.1. 文書属性定義

`src/docs/asciidoc/attribute.adoc` では文書のデフォルトキャプション名などの属性定義が行えます。画像挿入時の「図.」など標準で付与されるキャプション文字列が定義できます。必要に応じて修正してください。

`src/docs/asciidoc/attribute.adoc`

```
..snip..
:preface-title: まえがき
:toc-title: 目次
:appendix-caption: 付録
:caution-caption: 注意
:example-caption: 例
:figure-caption: 図
..snip..
```

Asciidoctor で利用可能な属性は次から参照できます。

<https://asciidoctor.org/docs/user-manual/#attributes>

Attributes are one of the features that sets Asciidoctor apart from other lightweight markup languages. Attributes can activate features (behaviors, styles, integrations, etc) or hold replacement (i.e., variable) content.

— asciidoctor.org

2.4.2. PDF テーマ定義

`src/docs/asciidoc/@style/pdf-theme.yml` では PDF 文書のスタイルを定義できます。PDF 文書の紙サイズやマージン、フォント・行送り幅などが設定できます。必要に応じて修正してください。

属性の詳細は次から参照できます。

<https://docs.asciidoctor.org/pdf-converter/latest/>

Asciidoctor PDF Theming Guide

— asciidoctor.org

`pdf-theme.yml` の修正は `.adoc` 修正時のビルドで PDF 文書に反映します。`pdf-theme.yml` の修正だけを反映したい場合は、いずれかの `.adoc` 文書に影響のない修正を加えてファイルを更新するか、`./gradlew clean` してからビルドを実行してください。



本手順で採用している Asciidoctor PDF のバージョンは 1.6.2 です。日本語禁則処理を正しく動作させるために、現在のところ最新の 2.3 にはなっていません。新しい Theming Guide ドキュメントの属性や動作とは乖離している場合があることに注意してください。

2.5. 執筆の開始

一通りのサンプル文書の構成確認が終りましたら、次のような流れでご自身の執筆を開始してみてください。

1. サンプル文書の `src/docs/asciidoc/Cahpter02` 以降のフォルダの削除を行い見通しを良くする。
2. `src/docs/asciidoc/index.adoc` からの 1. で削除した `Chapter*` の `include` を削除する。
3. `src/docs/asciidoc/index.adoc` で文書のタイトルや版数などを設定する。
4. `src/docs/asciidoc/Cahpter00/index.adoc` を空にして目次前に挿入される「はじめに」や「本書の読み方」等の執筆。
5. `src/docs/asciidoc/Cahpter01/index.adoc` を空にして本編の執筆を開始。
6. 事前に目次案がある場合は、このタイミングで `Chapter*` フォルダと対応する `index.adoc` を新規作成し、目次案にあった章や節などの大項目のみを準備するのも良い方法です。目次が完成し全体の把握がしやすくなります。
7. ここで一度 `./gradlew docs` を行い、想定通りの HTML/PDF 文書が `docs/` 内に生成されていることを確認する。外観の修正点があれば、
 - 文書形式の場合は `src/docs/asciidoc/attribute.adoc` を調整。
 - PDF テーマの場合は `src/docs/asciidoc/@style/pdf-theme.yml` を調整。
 - HTML テーマの場合は `src/docs/asciidoc/@style/asciidoctor.css` を調整。

2.6. 執筆のイテレーション

ここまで手順で執筆環境が整った後の、文書執筆の進め方のイメージは次のようになります。

1. プロジェクトフォルダを VS Code で開く。
2. `src/docs/asciidoocs` 配下の Asciidoc 文書を VS Code でプレビューしながら執筆する。
 - `Chapter` が増えた場合は `src/docs/asciidoocs/index.adoc` に `Chapter` を追記し、`src/docs/asciidoocs/Chapter*/index.adoc` ファイルを新規追加する。
 - 画像ファイルが必要な場合は、`src/docs/asciidoocs/Chapter*/images` フォルダに追加する。
3. 文書執筆の一定の区切りをもって `./gradlew docs` で文書のビルドを行い、HTML/PDF 文書を `docs/index.html` と `docs/index.pdf` で推敲する。
 - `./gradlew docs` の入力は VS Code の統合ターミナルが使えます。
4. 必要ならばプロジェクトディレクトリを Git などのバージョン管理下として、コミットを行う。
5. 本日の執筆を終え、明日はまた 1. に戻る。

文書の推敲

文書の推敲は、iPadなどのペンシルデバイスを持つ機械に PDF 文書ファイルを転送して「赤入れ」すると便利です。さらに、Dropbox や OneDrive などのクラウドストレージにファイルを配置すると執筆環境と推敲情報を共有できます。



画面. iPad の Goodnotes アプリによる PDF 文書の推敲

2.7. 文書の Git 管理

執筆中の文書をインターネット上に配置した Git リポジトリで管理すると次のようなメリットがあります。

- 文書の修正差分、版数管理ができる。
- 信頼できる場所にバックアップ・情報保全がされる。
- クラウド環境での執筆が可能になる。
- 複数環境での執筆、及び共同執筆時にファイルの混乱がなくなる。
- GitHub Actions などの CI 機能を使い継続的な文書の公開ができる。

本項では Git リポジトリとして GitHub サービスを使うことを例に、文書のプロジェクトファイルを Git 管理する手順を示します。

2.7.1. Git クライアントの導入

お使いの PC に Git クライアントが未導入の場合は、次の手順でインストールします。

macOS の場合(ターミナルエミュレータで以下のコマンドを実行)

```
brew install git
```

Ubuntu の場合(ターミナルエミュレータで以下のコマンドを実行)

```
sudo apt install git
```

Windows の場合(PowerShell ウィンドウで以下のコマンドを実行)

```
winget install --id Git.Git -e --source winget
```

Git クライアントの導入後、次のドキュメントを参考に GitHub アカウントへの公開鍵設定を行ってください。

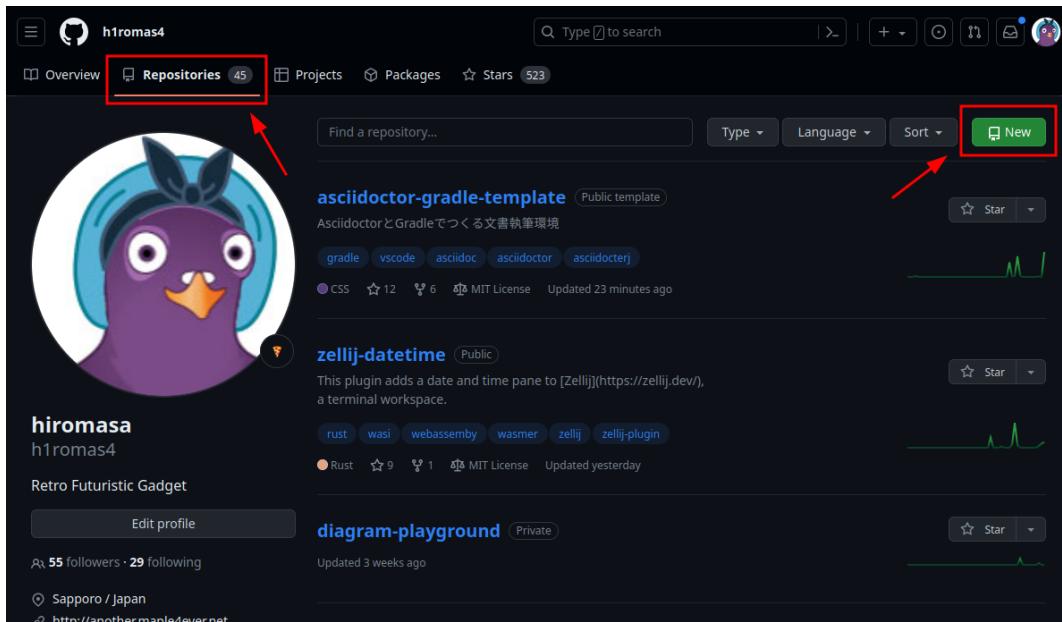
<https://docs.github.com/ja/authentication/connecting-to-github-with-ssh/adding-a-new-ssh-key-to-your-github-account>

新しい(または既存の) SSH キーを使うように GitHub.com 上のアカウントを構成するには、アカウントにキーを追加する必要があります。

— GitHub アカウントへの新しい SSH キーの追加

2.7.2. GitHub 上に文書用のリポジトリを作成

GitHub にログインし、各自のホーム画面から **Repositories > New** を押下します。

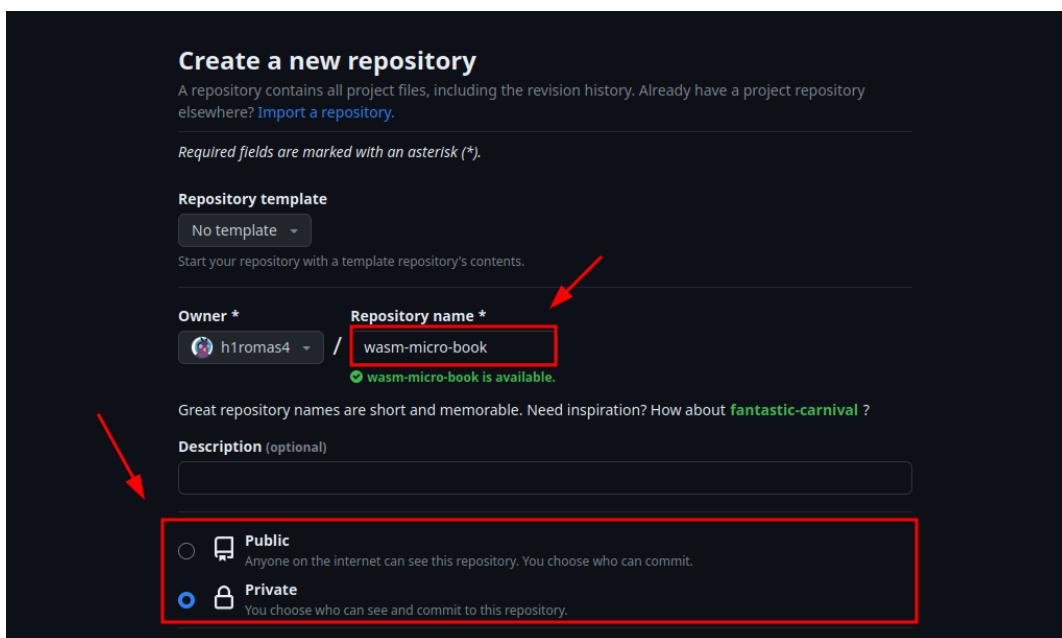


画面. GitHub の Repositories

Create a new repository 画面で Repository name を入力し、リポジトリが公開 (Public) か非公開 (Private) かを選択し、最後にページ下にある [Create repository] を押下します。

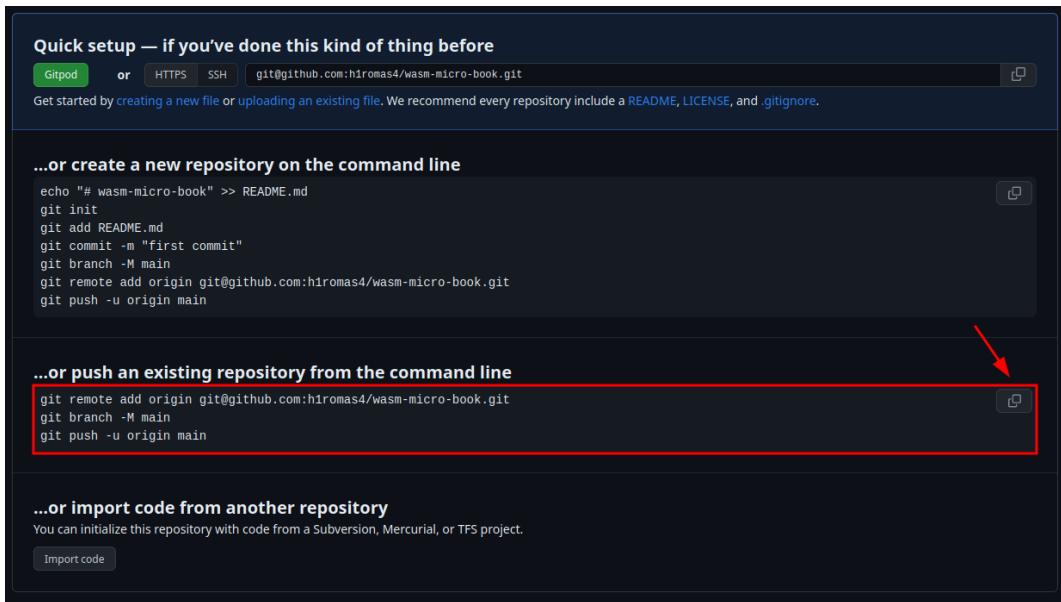


ここではリポジトリ名として `wasm-micro-book` を、公開範囲を執筆者のみが文書ファイルが閲覧・操作が可能となる `Private` を選択しました。なお、公開範囲に `Public` を指定した場合は、インターネット上の全てのユーザから文書ファイルの読み取りが可能になりますので注意してください。



画面. GitHub の Create a new repository

リポジトリの作成が終わると画面が遷移して "…or create a new repository on the command line" 項が現れます。次項の手順で必要になりますので、表示されたコマンドをクリップボード等に保持しておきます。



画面. GitHub の Quick setup

2.7.3. GitHub 上に文書を初期プッシュ

GitHub リポジトリに文書を配置するため、VS Code で PC 上の文書が配置されているプロジェクトフォルダを開き、統合ターミナルから以下のコマンドでローカルの Git リポジトリを初期化します。



プロジェクトフォルダは、ルートに gradlew などの本手順のビルドスクリプトが配置されている構成になっていることを確認してください。変換スクリプトやフォルダ構成ごと Git にコミットすることで、いずれの執筆環境でも HTML/PDF 変換処理ができるようになります。

VS Code 統合ターミナルから Git リポジトリを初期化

```
git init  
git config --local user.name h1romas4 ①  
git config --local user.email h1romas4@gmail.com ②  
git add .  
git commit -m 'initial commit'
```

① h1romas4 任意の名前に変更。Git コミットログの記載される氏名になります。

② h1romas4@gmail.com 同上。Git コミットログに記載されるメールアドレスになります。

ローカルの Git リポジトリの初期化完了後、続けて統合ターミナルで前項「GitHub の Create a new repository」画面でクリップボード等に保持したコマンドを実行し、GitHub リモートリポジトリにプロジェクトファイル一式をプッシュ操作します。

VS Code 統合ターミナルから GitHub にプッシュ

```
git remote add origin git@github.com:h1romas4/wasm-micro-book.git ①
git branch -M main
git push -u origin main
```

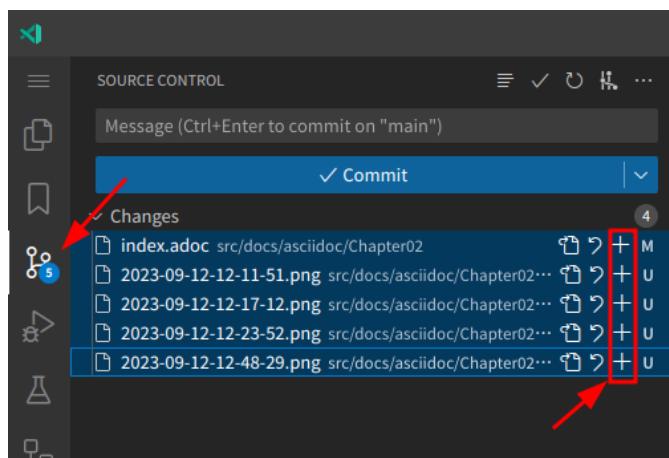
① h1romas4/wasm-micro-book.git 部分が前項で作成したリポジトリ名になります。

以上で GitHub リポジトリへの文書の配置が完了しました。ウェブブラウザーで GitHub リポジトリにアクセスしすると、プッシュした文書ファイルがあることが確認できるでしょう。

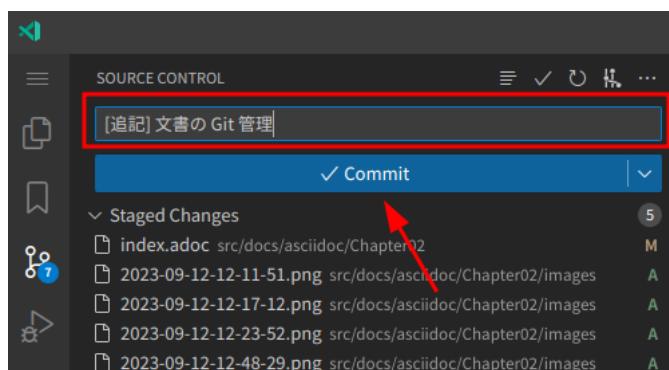
2.7.4. 修正したファイルのコミット・プッシュ

Git によるファイル管理を始めた後、VS Code 上で文書の編集を行うと修正や追加のあったファイルが左のサイドバーに通知されます。

この修正を Git に反映させるために、[+] で Git 上にステージングし、任意のコミットコメントを記載して [Commit] を押下するコミット操作を行います。



画面. VS Code のファイル修正通知と Git へのステージング

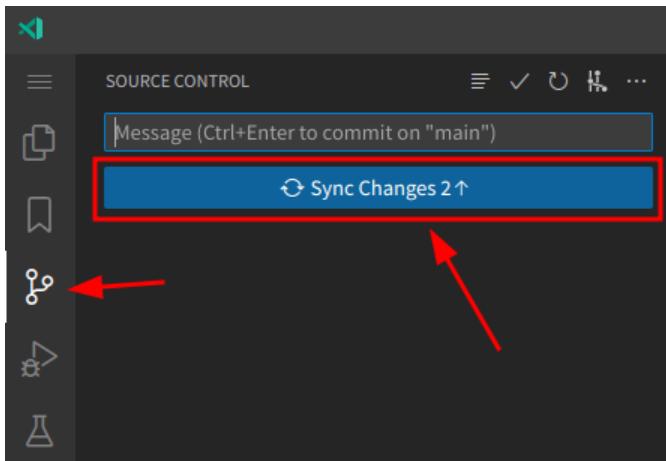


画面. VS Code から Git へのコミット

コミットはローカル上の Git に対して行われます。この後 GitHub のリモートリポジトリに反映させるため [Sync Changes] を押下しプッシュ操作を行います。



プッシュ操作を行うまで GitHub リモートリポジトリには文書の修正が反映されません。忘れずに実施してください。操作に慣れるまでは、プッシュ操作後にウェブブラウザで GitHub リポジトリにアクセスしファイルの更新を確認すると良いでしょう。



画面. VS Code から GitHub Repository へのプッシュ

なお、[Sync Changes] ボタン押下操作の他、統合ターミナルから `git push` コマンドを入力することでも同様の結果が得られます。

統合ターミナルから `git push` を実行

```
git push
```

2.7.5. 異なる執筆環境での Git 操作

異なる執筆環境から GitHub リポジトリの文書を取得するには次の操作を行います。

1. GitHub 上のリポジトリをクローンしプロジェクトフォルダを得る。 (初回のみ操作)

```
git clone git@github.com:h1romas4/wasm-micro-book.git ①
```

① `h1romas4/wasm-micro-book.git` 部分が前項で作成したリポジトリ名になります。

2. 別の執筆環境からの修正（プッシュ）を取得するために `git pull` 操作を実施。

統合ターミナルから `git pull` を実行

```
git pull
```

個人の執筆活動の Git 操作としては「執筆開始前に `git pull`」「執筆区切りで `commit` と `push`」と覚えておくだけで十分機能するはずです。



もし共同著者間での文書マージ操作等、より高度な Git の操作が必要になってきた場合は、ウェブサイトや書籍から Git についての優れたドキュメントを得られます。

2.8. クラウド環境による執筆

クラウド開発環境である [Gitpod](#) や GitHub の [Codespaces](#) を活用すると、ローカル環境の準備なしにウェブブラウザだけで執筆と HTML/PDF 文書変換が可能です。このことから、サブマシンや iPad などタブレット端末でも執筆活動が行えます。

<https://www.gitpod.io/>

The developer platform for on-demand cloud development environments. Create software faster and more securely.

— Gitpod

<https://github.co.jp/features/codespaces>

超高速で構築するクラウド開発環境

— Codespaces | GitHub

2.8.1. Gitpod

Gitpod クラウド上での執筆にあたっての条件は次のとおりです。

- 文書ファイル一式を GitHub、GitLab、Bitbucket のいずれかに配置する必要があります。（プライベートリポジトリも利用可能です）
- 2023-07 現在 Gitpod の無料枠は 50時間/月です。最新情報は <https://www.gitpod.io/pricing> から確認できます。

準備ができれば、次の要領で URL を作成しウェブブラウザでアクセスするとクラウド環境が起動します。

<https://gitpod.io/new/#https://github.com/h1romas4/asciidoc-gradle-template> ①

- ① URL の先頭に <https://gitpod.io/new/#> という Gitpod の URL を前置し、その後の <https://github.com/h1romas4/asciidoc-gradle-template> 部分に自身の GitHub リポジトリ URL を付与してウェブブラウザの URL 欄に貼り付けます。



上記記載通りの URL にアクセスすると本手順のリポジトリで Gitpod クラウドを試用できます。なお、リポジトリには筆者以外の書き込み権がついていないため Git への文書修正操作は試せません。

The screenshot shows a browser window with two tabs: 'index.adoc - asciidoctor.io' and 'Preview index.adoc'. The left pane displays the Asciidoc source code, which includes sections like 'Introduction', 'Preface', and a note about using AsciiDoctor or AsciiDoc. The right pane shows the rendered HTML output with headings and text from the source. Below the browser is a status bar with information like 'Ln 16, Col 132, Spaces: 4, UTF-8, LF, AsciiDoc, Layout: us, Ports: 33191, 39889, 42693, 44897'.

```

src > docs > asciidoc > Chapter00 > index.adoc
>はじめに
1 include:../attribute.adoc[ ]
2
3 [[introduction]]
4 [preface]
5 == はじめに
6
7 本文書は ``Asciidoc``、その Ruby による実装である ``Asciidoctor`` を用いて AsciiDoctor 文書を執筆する環境を構築する手順を示します。実行環境は Windows、Linux、macOS の各 OS に対応しています。
8
9 この文書の手順により以下のことができるようになります。
10
11 AsciiDoc 形式で執筆した文書を HTML/PDF 形式に変換:
12 | AsciiDoc 文書変換用 Gradle スクリプト
13 | テキストエディターで変換結果をリアルタイムでプレビューしながら文書を編集:
14 | Visual Studio Code と AsciiDoc 拡張設定
15
16 Asciidoc は表現力の高い文書をテキストファイルベースで執筆できるテキストプロセッサーです。他の軽量テキストプロセッサーが持たない文書間のインクルードやソースコードの挿入などの機能も有し、かつ簡単です。特に技術文書の執筆には大きな力を発揮することでしょう。
17
18 [TIP]
19 -----
20 AsciiDoc の表現力を示すひとつの例は、このような脚注表現です。
21 ====
22
23 一般的にこのようなテキストプロセッサーを用いた執筆環境を構築するためには多くの準備が必要となります。本文書の手順は極力初期導入するプロダクトを少なく、簡単に快適な執筆環境を整えられるよう考案されました。
24
25 具体的には文書の変換に、実行を JVM 環境だけに依存する ``AsciidoctorJ`` と ``Gradle`` を活用し、執筆環境については ``Visual Studio Code`` を用いることでリアルタイムで文書をプレビューしながら、最後にコマンド一つで HTML/PDF 化できるように準備してあります。PDF 出力に関しては、フォントセットや禁則処理設定をプロジェクトにもたせることにより、実行環境に問わらず同一出力が得られるようになっています。
26
27 Asciidoc による文書の執筆は形式的効率が良く、また更新差分が取りやすいため文書履歴管理や共同執筆環境としても有効です。このメリットをさらに活用するため、本手順ではテキストダイアグラム記法による図形や数式のベクター画像挿入もサポートしました。

```

画像.ウェブブラウザ上で起動した VS Code と Asciidoc 拡張(Gitpod)

本手順向けの VS Code 用の推奨拡張が自動導入されるように設定されていますので、特に操作をせずそのまま文書の執筆が開始できます。

また合わせて、統合ターミナルから HTML/PDF のビルドができるようコンテナ環境も自動設定されます。ターミナルに以下の表示が出力されれば準備完了です。同ターミナルで `./gradlew docs` することによりローカル環境と同様に HTML/PDF の文書変換が開始されます。

```

#
# asciidoctor-gradle-template
#
# Setup Done! Build document here. (./gradlew docs)
#
# It will build correctly, if the following conditions 'OK'
#
# Java(openjdk): OK
# Dialog Font(Noto Sans CJK JP): OK
# LANG(ja_JP.UTF-8): OK
#

```



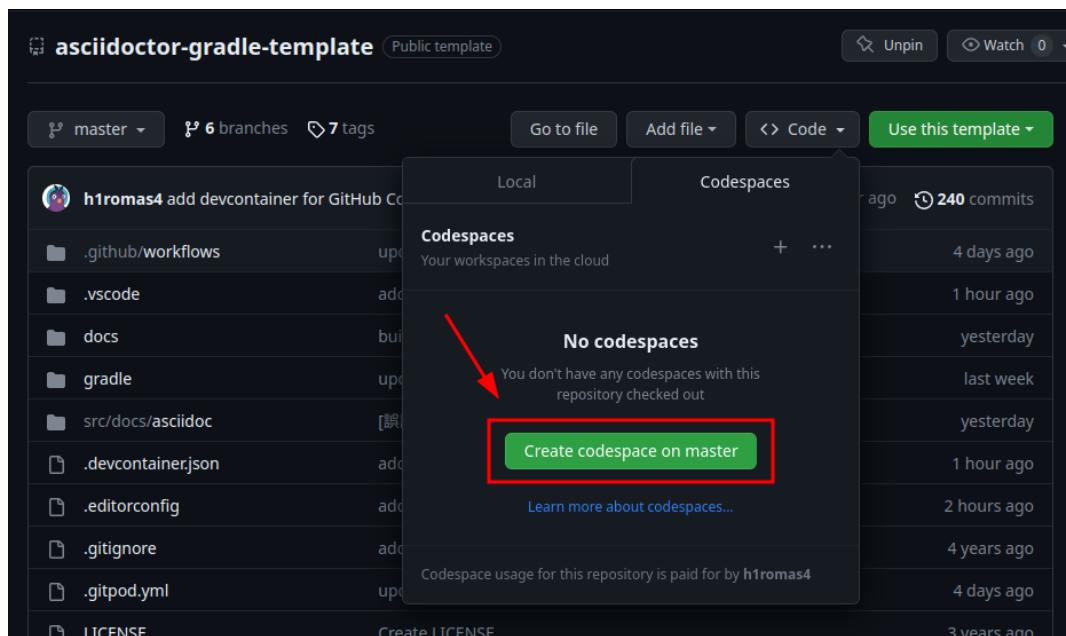
これらの Gitpod 向けの設定は `.gitpod.yml` ファイルで定義されています。

2.8.2. Codespaces

Codespaces クラウド上での執筆にあたっての条件は次のとおりです。

- 文書ファイル一式を GitHub に配置する必要があります。（プライベートリポジトリも利用可能です）
- 2023-07 現在 Codespaces の無料枠はおおよそ 60時間/月ほどです。最新情報は各自の GitHub アカウント <https://github.com/settings/billing> の Codespaces 項から確認できます。

準備ができれば、GitHub サイトにログインした上でファイル一式をコミットしたリポジトリを開き **Code > Codespaces > Create codespace on ブランチ名** を押下します。



i <https://github.com/h1romas4/asciidoc-gradle-template> にアクセス後、上記の操作を行うと、本手順のリポジトリで Codespaces クラウドを試用できます。なお、リポジトリには筆者以外の書き込み権がついていないため Git への文書修正操作は試せません。

2. Asciidoc から HTML/PDF 文書を作成する

ボタン押下後コンテナのビルドに画面が遷移し、完了するとウェブブラウザ上で VS Code が起動します。



画像.ウェブブラウザ上で起動した VS Code と Asciidoc 拡張(Codespaces)

前項の Gitpod と同様に VS Code の拡張は自動導入され、PDF/HTML ビルド用のコンテナ環境も構築されます。コンテナ環境の構築中は統合ターミナルに以下のような表示がされ、終了とともに自動的に閉じられます。

```
Use Cmd/Ctrl + Shift + P -> View Creation Log to see full logs
✓ Finishing up...
  Running postCreateCommand...
> bash .devcontainer/build-codespaces-contaner.sh
```

PDF/HTML のビルドは本処理が終了後、新しい統合ターミナルを開き環境が日本語設定になっていることを確認の上 `./gradlew docs` を実行します。

```
$ echo $LANG
ja_JP.UTF-8 ①
$ fc-match Dialog
NotoSansCJK-Regular.ttc: "Noto Sans CJK JP" "Regular" ②
$ ./gradlew docs ③
# .snip..
BUILD SUCCESSFUL in 3m 19s
```

① `ja_JP.UTF-8` になっていること

② `Noto Sans CJK JP` になっていること

③ 文書のビルドを開始

統合ターミナルの環境が日本語設定になっていない場合、文書中のダイアログ記法で生成された図表の日本語出力が崩れる場合があります。



クラウド環境初期後に自動的に開かれる統合ターミナルは日本語設定が反映していないため、文書のビルトを行う場合は混乱がないように `exit` コマンドで一旦閉じた上で再起動すると良いでしょう。



Codespaces 向けの設定は `.devcontainer/devcontainer.json` 及び `.devcontainer/build-codespaces-contaner.sh` ファイルで定義されています。

2.9. クラウド環境でのファイル保存と執筆の再開

クライド環境での文書ファイルの保存は、基本的に Git のコミット・プッシュ操作になります。通常の [Ctrl + s] のファイル保存操作とともに、一区切りの間隔で Git のコミットとプッシュの操作を行うのを忘れないようしてください。



Git プッシュの操作を行わずにウェブブラウザを閉じてしまった場合はファイルが失われる可能性があります。十分注意をしてください。

とは言ってもウェブブラウザのタブは閉じやすいもの。もしうっかり閉じてしまった場合は、GitHub にログインした状態で以下の URL にアクセスしてください。

Gitpod の場合

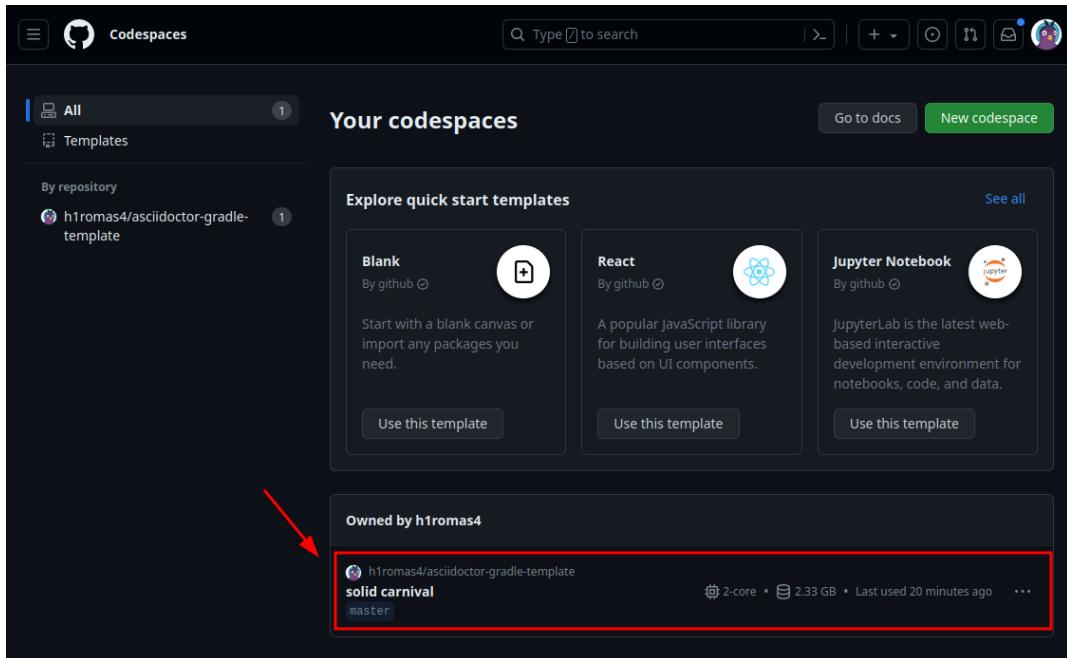
<https://gitpod.io/workspaces>

The screenshot shows the Gitpod interface with the title "Workspaces". It displays a list of recent workspaces. One workspace, "h1romas4-asciidoc", is highlighted with a red box. The workspace details shown are:

- Name: h1romas4-asciidoc...
- Repository: h1romas4/asciidoc-gradle-t...
- Branch: master
- Last Commit: 17 minutes ago
- Changes: No Changes
- More options: three dots (...)

Codespaces の場合

<https://github.com/codespaces>



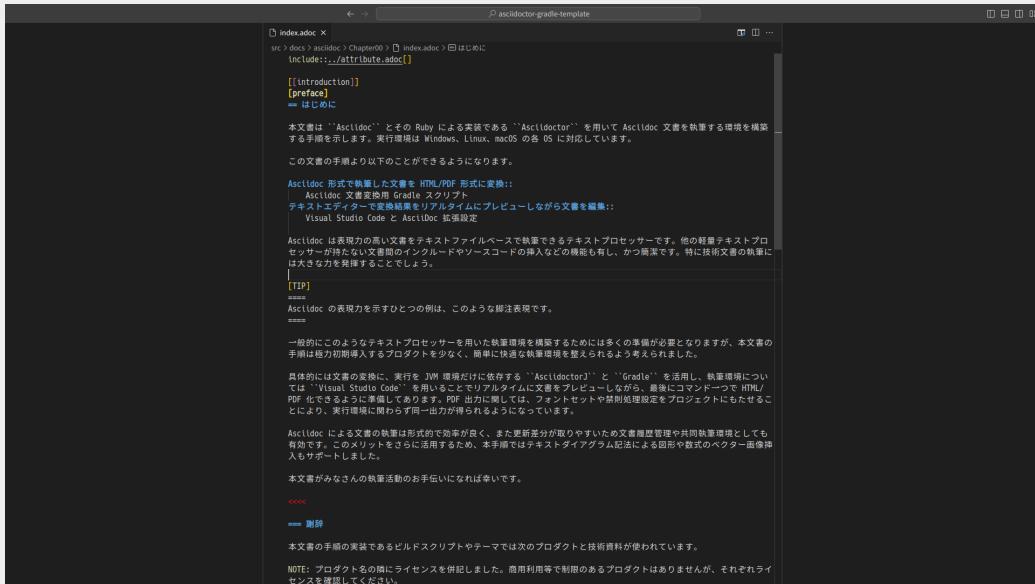
クラウド上で起動していた環境は最終利用時からある一定期間（数日）は残ります。この動作を利用して未コミット・プッシュのファイルを取り戻せます。

一覧画面にはこれまで使っていた環境が表示されますので、選択して「開く」操作をしてください。どちらも右の三点リーダメニューより、Gitpod では [Open]、Codespaces では Open in > Open in browser を押下すると、未プッシュのファイルとともに環境が復元します。

また、コミット・プッシュ済みに関わらず、翌日執筆を再開する際もこれらの画面から環境をリオープンします。クラウドのリソース削減になるとともに、これまでの VS Code の操作状態が維持されているため時間の短縮にもなります。

VS Code の Zen Mode

VS Code には執筆に集中したい場合に、不要な UI を隠しながらフルスクリーン表示をする Zen Mode があります。標準のキーアサインでは [Ctrl + k, z] でモードが遷移し、再度同じ操作で元の画面に戻ります。



Zen Mode は、VS Code や OS の不要なコンポーネントが見えなくなり画面が広く使えること、また思考を遮る要素がなくなることから快適な執筆をする上で良いツールとなることでしょう。

2.10. GitHub Actions によるビルド

本手順のリポジトリに含まれるファイル `.github/workflows/gradle.yml` 及び `.github/workflows/release.yml` は GitHub サーバ上で文書の変換テスト、及び `.tar.gz` 形式での HTML/PDF 文書のリリースを行うための定義ファイルです。

これらのファイルを文書ファイルとともに各自のプロジェクト GitHub リポジトリにコミットすることで、文書変換とファイルリリースが git の操作を契機として GitHub サーバ上で自動的に動作するようになります。



本項「GitHub Actions によるビルド」機能の利用は必須ではありません。大人数で継続的に執筆を行いたいケースなどに採用してみてください。HTML/PDF 変換処理の自動化やファイルリリースの版数管理で大きな力を発揮することでしょう。

2.10.1. `.github/workflows/gradle.yml`

git による main ブランチへの push 操作から自動的に動作する文書変換テストです。Linux、Windows、macOS のそれぞれの環境で、Java 11 と 17 においてビルドが試行され、一定の検査が行われます。

環境はマトリクスになっており、計 6 環境でのテストでビルドが正しいかが確認され GitHub Actions 上のログやメール通知で結果が得られます。自分の執筆環境以外のさまざまな環境で、変換処理が実行できることへの確信が持てるはずです。

検査に関しては、文書が生成されたかを基本として確認を行い、文書の内容自体に対しては行われません。

- ・ HTML、PDF 文書ファイルが存在するか。
- ・ 後述の asciidoctor-diagram による `.svg` 画像生成が動作したか。
- ・ GitHub Actions ログへのファイル数、一覧の出力。

検査は以下の部分で行われます。文書に合わせたさらなる検査が必要な場合はシェルスクリプトでカスタマイズしてください。

`.github/workflows/gradle.yml`

```
- name: Check
  shell: bash
  run: |
    find docs/ -type f -ls
    find docs/ -type f -ls | wc
    # test asciidoctor-diagram (If you are not using diagram, comment out the next)
    find docs/ -type f -ls | egrep 'svg$' ①
    # test asciidoctor
    file docs/index.pdf | grep PDF
    file docs/index.html | grep HTML
```

① asciidoctor-diagram を使った `.svg` 図表が生成されているかの検査。文書中で使用していない場合は削除するか、コメントアウトしてください。

また次の部分で PDF 文書のメタデータの出力を行うサンプルを組み込んでいます。PDF 文書に不要なメタデータが含まれていないか等の確認が行えます。



この部分は `pdfinfo` コマンドを活用している関係で Ubuntu ビルドのみ対応しています。
また、合否確認は組み込んでいませんので必要であればカスタマイズしてください。

```
- name: Output PDF information
if: startsWith(matrix.os, 'ubuntu')
run:
  echo "# PDF information"
  pdfinfo docs/index.pdf
  echo "# URLs in document"
  pdfinfo -url docs/index.pdf | tail -n +2 | awk '{ print $3 }' | sort | uniq
```

なお、文書が問題なく変換ができたことの証となる「バッジ」は以下のようにすることで画像として取得可能です。正常の場合はグリーンに、失敗の場合は赤で表示されます。

```
https://github.com/\[ユーザ名\]/\[リポジトリ名\]/workflows/Build/badge.svg
```

具体的には、本手順のリポジトリの GitHub 向けの `README.adoc` は次のようにになっています。

```
image::https://github.com/h1romas4/asciidoc-gradle-template/workflows/Build/badge.svg[]
```



2.10.2. .github/workflows/release.yml

git のバージョンタグを push する操作で、GitHub のリリースページにソースコード一式と、GitHub Actions 上で変換処理された `docs` 配下のファイルを `{doc-name}-{verison}-by-*.`tar.gz アーカイブファイルとして GitHub のリリースダウンロードページに掲載する GitHub Actions 定義です。

ファイル名の `by-*` 以下の部分にはビルドした環境名が `by-windows-latest-java-11` や `by-macos-latest-java-17` のような形で付与されます。通常はひとつの環境でビルドした文書だけリリースできれば十分ですので、利用する時は次のマトリクスを変更してビルド環境を絞ってください。

```
jobs:
  build:
    strategy:
      matrix:
        os: [ ubuntu-22.04, windows-latest, macos-latest ] ①
        java: [ '11', '17' ] ②
```

① os: [windows-latest] とすると Windows のみでビルドされます。

② java: ['11'] とすると Java 11 のみでビルドされます。

設定する OS や Java のバージョンは、メインで使われている執筆環境と合わせると良いでしょう。



本文書はテンプレートとして多くの環境での動作を担保するため、Ubuntu | Windows | macOS 上のそれぞれの Java 11 | 17 の計 6 環境で文書のビルドと確認を実施しています。

リリースされる文書のファイル名やバージョンについては、スクリプトの次の部分で指定されています。執筆する文書に合わせて変更してください。

```
- name: Set archive attribute
  id: get-attr
  shell: bash
  run: |
    # document filename
    echo "doc-name=asciidoc-gradle-book" >> $GITHUB_OUTPUT ①
    # document version from git tag
    echo "version=${GITHUB_REF#refs/*/}" >> $GITHUB_OUTPUT ②
```

① asciidoctor-gradle-book 部分が出力されるファイル名に前置されます。

② Git のタグ名がそのままバージョン名となります。

本リリースの機能を使い頒布や出版時にタグを打つことで、その時点のファイル断面がダウンロード可能な形で永続化できます。文書の版管理にご活用ください。

3. Asciidoc 記法

ここでは Asciidoc 文書で利用可能な記法の中で、特に技術文書で使いそうなものを取り上げて解説します。

Asciidoc 記法の入力の助けになるよう、プロジェクトフォルダの ``.vscode/asciidoc.code-snippets 設定には VS Code の [Ctrl + Space] 操作で補完可能な形式で Asciidoc 記法のスニペットが格納されています。プロジェクトで定形の記法があれば追加・修正すると便利でしょう。

```

8 Asciidoc 記法の入力の助けになるよう、プロジェクトフォルダの ``.vscode/
9
10 ag-
11   □ ag-button          asciidoc-button
12   □ ag-caution         asciidoc-caution
13   ima □ ag-caution-block asciidoc-caution-block
14   □ ag-column          asciidoc-column
15   == □ ag-command       asciidoc-command
16   □ ag-comment         asciidoc-comment
17   文書 □ ag-h2          asciidoc-h2
18   □ ag-h3              asciidoc-h3
19   [so] □ ag-hr          asciidoc-hr
20   --- □ ag-image        asciidoc-image
21   [[p □ ag-image-with-caption asciidoc-image-with-caption
22   == □ ag-important      asciidoc-important
23
24 [[introduction]] // <2>
25 == はじめに
26

```



補完文字列は他の補完と区別しやすいよう ag- が先頭に付与されています。補完操作後、最初に ag- と入力することで一覧できます。

3.1. 文書中の相互参照

文書内の参照リンクは .adoc 文書中で次のように指定します。HTML/PDF でクリックリンクが生成されます。

```

[[project-structure]] ①
== Asciidoc 記法

[[introduction]] ②
== はじめに

<<project-structure,章の冒頭>> ①
<<Chapter01/index.adoc#introduction,はじめに>> ②

```

① 同一 .adoc 内での内部参照

② .adoc をまたいだドキュメント間参照

リンクの例

- 章の冒頭 を参照してください…

- [はじめに](#) で述べたように…

3.2. 外部リンク

インターネット上の URL は自動的にリンクに変換されます。

<https://h1romas4.github.io/asciidoc-gradle-template/index.html>

次のように URL の末尾に [] を定義すると指定した文字列からのリンクとなります。

[https://h1romas4.github.io/asciidoc-gradle-template/index.html\[AsciidocとGradleでつくる文書執筆環境\]](https://h1romas4.github.io/asciidoc-gradle-template/index.html[AsciidocとGradleでつくる文書執筆環境])

AsciidocとGradleでつくる文書執筆環境



PDF の印刷を考慮する場合は、文字列リンクを用いると URL の情報が消えてしまうため、次項の引用機能を用いると良いでしょう。

3.3. 引用

外部の文献を参照する引用は次のように記述します。執筆している文書が紙面となり、引用先がウェブサイトの場合はいずれかの位置に URL を掲載します。

[quote, Asciidoc Documentation Site]

<https://docs.asciidoc.org/>

Welcome to the Asciidoc documentation site! Here you can find the reference material, guides, and examples to write content in AsciiDoc and publish it using Asciidoc. This documentation will help you start your journey with AsciiDoc or dive deeper if you're already well on your way.

<https://docs.asciidoc.org/>

Welcome to the Asciidoc documentation site! Here you can find the reference material, guides, and examples to write content in AsciiDoc and publish it using Asciidoc. This documentation will help you start your journey with AsciiDoc or dive deeper if you're already well on your way.

— Asciidoc Documentation Site

3.4. 画像

画像の挿入は、画像ファイルを編集中の .adoc 文書から相対パスでみた images/ フォルダに格納した上で、次のように記述します。

```
image::2023-06-30-12-11-56.png[pdfwidth=70%, width=600px]
```

pdfwidth 属性では PDF 紙面に対する幅の比率を、width 属性では HTML 文書中の幅の比率もしくはピクセル値を指定します。

なお、本手順の Asciidoc 文書の構成では images に格納されている画像ファイルパスに images/ を付与する必要はありません。



これは src/docs/asciidoc/attribute.adoc で :imagesdir: ./images が設定済みであるための動作です。

3.5. ソースコードシンタックスハイライト

[source] ブロックを用いて文書中にソースコードをシンタックスハイライト付きで埋め込みます。また、ソースコードに // <1> などのコメント形式でソースコード下部に対応する説明が記述できます。

```
[source, javascript]
[caption="""]
JavaScript ソースコードのシンタックスハイライトの例
-----
function hello() { // <1>
    console.log("Hello, World!"); // <2>
}
-----
<1> 関数定義
<2> コンソールに ``Hello, World`` を出力
```

JavaScript ソースコードのシンタックスハイライトの例

```
function hello() { ①
    console.log("Hello, World!"); ②
}
```

① 関数定義

② コンソールに Hello, World を出力

3.6. インラインコマンド・ファイル名

文中に現れるコマンドやファイル名などは `コマンド` で記述します。

```
``コマンド``
```

3.7. メニュー・ボタン

コンピュータの操作で使われる `メニュー > File > Open` や `[OK]` ボタンなどは次のように記述します。

```
menu: メニュー[File > Open]
```

```
btn:[OK] ボタン
```

3.8. ファイルインクルード

.adoc ファイルから別のファイルをインクルードできます。ソースコードや後述のダイアグラム形式などを別ファイルとして切り出すのに便利です。



Gradle のファイル更新監視は .adoc からインクルードされたファイルまでは及びません。インクルード先のファイルだけを書き換えた場合は ./gradlew docs が処理なしで終了してしまいますので、この場合は ./gradlew clean するなどしてからリビルドしてください。

```
[source, rust]
-----
include::source/hello.rs[]
-----
```

インクルードにより挿入されたソースコードの例

```
fn main() {
    println!("Hello, world!");
}
```

3.9. 表形式

表形式は次のようなコードで定義します。データ形式、罫線表現についていくつかのオプションが設定できます。



属性の `cols="1,2"` ではカラムの比率を指定します。

表形式の例1(CSV 形式、ヘッダーなし、グリッドなし)

```
[format="csv",cols="1,2"]
[frame="topbot",grid="none"]
[caption=""']
.表形式の例1
|=====
|キー1,バリュー1
|キー2,バリュー2
|キー3,バリュー3
|=====
```

キー1	バリュー1
キー2	バリュー2
キー3	バリュー3

キーワード

キー1	バリュー1
キー2	バリュー2
キー3	バリュー3

表形式の例2(デフォルト形式、ヘッダーあり、グリッドあり)

```
[cols="1,2", options="header"]
[frame="topbot"]
[caption=""']
.表形式の例2
|=====
|カラム名1
|カラム名2

|キー1
|バリュー1

|キー2
|バリュー2
|=====
```

カラム名1	カラム名2
キー1	バリュー1
キー2	バリュー2

また、`cols="1,2a"` のように a を付与したセルは Asciidoc 記法がそのまま反映します。

表形式の例3(Asciidoc 形式セル)

```
[cols="1,2a", options="header"]
[frame="topbot"]
=====
|通常カラム
|Asciidoc 形式付きカラム

|
. リスト1
. リスト2
|
. リスト1
. リスト2
=====
```

通常カラム	Asciidoc 形式付きカラム
.リスト1 .リスト2	<ol style="list-style-type: none"> 1. リスト1 2. リスト2

3.10. リスト

箇条書き表現は次のように記述します。



Asciidoc はリスト中にリスト継続マーク (+) を使うことでブロックのネストが可能です。手順を順序ありリストで表現し、順序やインデントを維持しながら画像や表形式などの挿入をします。

```
.
. 手順1
. 手順2
+
リスト継続1
+
image::2023-06-30-15-39-26.png[pdfwidth=60%, width=60%]
. 手順3
.. 箇条書きのネスト1
... 箇条書きのネスト2
*** 順序なしリスト1
*** 順序なしリスト2
```

1. 手順1

2. 手順2

リスト継続1

```
9
10  btn:[button-name]
11  □ button          asciidoc-button
12  □ caution         asciidoc-caution
13  □ caution-block   asciidoc-caution-block
14  □ column          asciidoc-column
15  □ command         asciidoc-command
16  □ h2               asciidoc-h2
17  □ h3               asciidoc-h3
18  □ image            asciidoc-image
19  □ image-with-caption asciidoc-image-with-caption
20  □ important        asciidoc-important
21  □ important-block  asciidoc-important-block
22  □ link-external    asciidoc-link-external
== Asciidoc 記法
```

3. 手順3

- a. 箇条書きのネスト1
 - i. 箇条書きのネスト2
 - 順序なしリスト1
 - 順序なしリスト2

3.11. 脚注

補足点を脚注するいくつかの表現があります。

NOTE: メモ

TIP: チップス

IMPORTANT: 重要

WARNING: 警告

CAUTION: 注意



メモ



チップス



重要



警告



注意

脚注が長文になる場合は次のようにブロックを用います。

[NOTE]

=====

改行を含む脚注です。

吾輩は脚注である。名前はまだ無い。

=====



改行を含む脚注です。

吾輩は脚注である。名前はまだ無い。

3.12. 改ページ・水平線

改ページは次の記述で挿入します。PDF 文書のみに効果があり HTML 文書には影響しません。

```
<<<
```

水平線は HTML/PDF 文書のどちらでも有効です。

```
- - -
```

3.13. コラム表現

Asciidoc のサイドバーとブロック名称機能を組み合わせ、コラム表現を挿入します。

```
. コラム
```

```
****
```

ブロックを使ってコラム表現を挿入します。

吾輩はコラムである。名前はまだ無い。

```
****
```

コラム

ブロックを使ってコラム表現を挿入します。

吾輩はコラムである。名前はまだ無い。

3.14. コメント行

成果物に現れない Asciidoc 文書へのコメントは次のように文頭に // で記述します。

```
// TODO: さらに Asciidoc 記法を追加していくこと。
```

4. ダイアグラム記法

本変換スクリプトでは `asciidocorj-diagram` が有効になっており、いくつかのダイアグラム記法を追加のアドイン無しで使えます。



PDF 出力時に日本語が化けないよう、自動的にフォントパッチが適用されるよう構成されています。

ダイアグラムを SVG ベクター画像で出力する指定は次のようにになります。

```
[ダイアグラム種類, 出力ファイル名, svg, pdfwidth=70%, width=480px]
```



出力ファイル名は `images` フォルダ内で一意になるように設定してください。特にダイアログ記法を別の場所からコピー・ペーストした場合は要チェックです。同名ファイル名が指定された場合は上書きされる動作になります。

`pdfwidth` は PDF 紙面に対しての比率指定、`width` は HTML 上の幅比率もしくはピクセル (px) で指定します。

本項では PlantUML と ditaa ダイアグラム記法を用いて、技術文書で使われやすい表現のいくつかの記述法を示します。

コピーして使いやすいよう各ダイアグラム内で頻出する記法をなるべくピックアップする形で描いています。作成したい図表と似たようなものからアレンジして使うと良いでしょう。

PlantUML と ditaa 記法の詳細な仕様については、次のドキュメントから参照できます。

<https://plantuml.com/ja/>

PlantUML in a nutshell

— PlantUML

<https://ditaa.sourceforge.net/>

DIagrams Through Ascii Art by Stathis Sideris

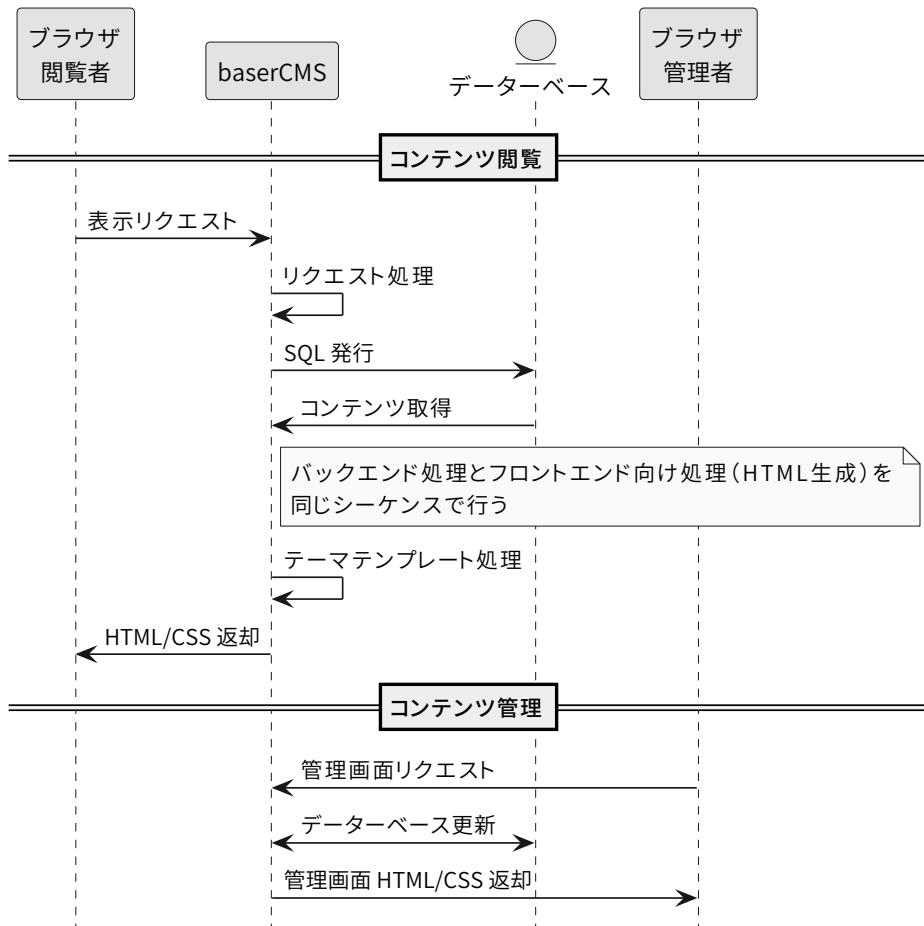
— ditaa

4.1. シーケンス図

シーケンス図を PlantUML ダイアグラム記法で記述します。

```
[plantuml, diag-sequence-sample2, svg, pdfwidth=70%, width=70%]
[caption=""]

-----
@startuml diag-sequence-sample2
skinparam monochrome true
hide footbox
participant "ブラウザ\n閲覧者" as browser
participant "baserCMS" as basercms
entity データベース as Entity
participant "ブラウザ\n管理者" as admin
== コンテンツ閲覧 ==
browser -> basercms: 表示リクエスト
basercms -> basercms: リクエスト処理
basercms -> Entity: SQL 発行
Entity -> basercms: コンテンツ取得
note right of basercms: バックエンド処理とフロントエンド向け処理 (HTML生成) を\n同じシーケンスで行う
basercms -> basercms: テーマテンプレート処理
basercms -> browser: HTML/CSS 返却
== コンテンツ管理 ==
admin -> basercms: 管理画面リクエスト
basercms <-> Entity: データベース更新
basercms -> admin: 管理画面 HTML/CSS 返却
@enduml
-----
```



PlantUML ダイアグラムのリアルタイムプレビュー編集

本手順の VS Code の推奨プラグインとなっている PlantUML 拡張 (jebbs.plantuml) を使うと、PlantUML のダイアグラム記法も VS Code 上でリアルタイムプレビューできます。

プレビュー処理をするために必要な PlantUML サーバは Docker コンテナで起動すると便利でしょう。なお、Gradle による文書の変換では内蔵の PlantUML ライブリリが使われますので本サーバーが起動している必要はありません。

```
docker run -d --rm -p 8080:8080 plantuml/plantuml-server:jetty
```

.vscode/setting.json は次のようにポート 8080 で PlantUML サーバが起動されることを期待して構成してあります。

```
.vscode/setting.json
```

```
{ "plantuml.server": "http://localhost:8080" }
```

文書で使う PlantUML ダイアグラムを .puml 拡張子で保存することで Asciidoc と同様にプレビューアイコンが表示され、プレビュー画面ではリアルタイムに .puml ファイルの修正が反映されます。

4.2. フォルダ・ファイルツリー

フォルダ・ファイルツリーを PlantUML の Salt 記法で記述します。

```
[plantuml, diag-salt-sample1, svg, pdfwidth=30%, width=280px]
[caption="""]

-----
' https://github.com/iconic/open-iconic/
@startsalt diag-salt-sample1
{
{T
+ <&folder> wp-admin
+ <&folder> wp-content
++ <&folder> themes
+++ <&folder> my-theme | 新規作成
++++ <&file> style.css | 新規作成
++++ <&file> index.php | 新規作成
++ <&folder> uploads
+ <&folder> wp-includes
+ <&file> .htaccess | 自動作成
+ <&file> index.php
}
}
@endsalt
-----
```



4.3. 数式

数式を PlantUML の AsciiMath 記法で記述します。

```
[plantuml, math-sample1, svg, pdfwidth=70%, width=70%]
[caption="""]

-----
@startmath math-sample1
f(t)=(a_0)/2 + sum_(n=1)^oo a_n cos((n\pi t)/L) + sum_(n=1)^oo b_n \ sin((n\pi t)/L)
@endmath
-----
```

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos\left(\frac{n\pi t}{L}\right) + \sum_{n=1}^{\infty} b_n \sin\left(\frac{n\pi t}{L}\right)$$

PlantUML の Salt 記法のフォントサイズ

Salt は現在のところ残念ながらフォントサイズの指定ができません。このため任意のフォントサイズにするためには出力画像の `pdfwidth` と `width` を設定して算出する必要があります。

```
@startsalt
<style>
saltDiagram {
    FontSize 10 ①
}
</style>
{
{↑
+ <&folder> wp-admin
}
}
@endsalt
```

① 未実装で効果がない

4.4. ブロック図

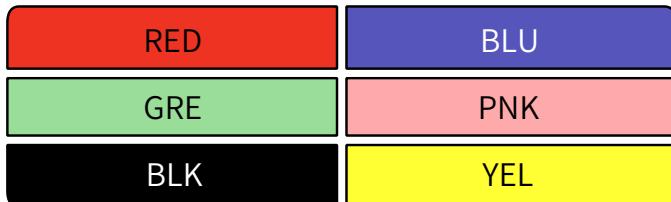
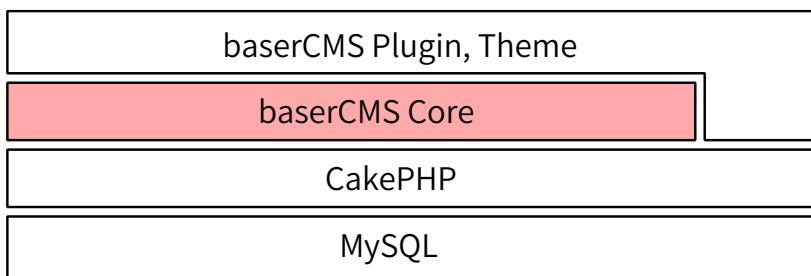
ブロック図を ditaa 記法で記述します。



ditaa 内で日本語を使う場合は、表示ずれを防ぐため半角スペースなどで文字数を調整する必要があります。また、ドロップシャドー shadows 設定は HTML 版に対してのみ有効で、現在のところ PDF 版に対しては効果がないようです。

```
[ditaa, diag-ditaa-block, svg, shadows=true, separation=true, pdfwidth=70%, width=70%]
[caption=""]

-----
+-----+
| baserCMS Plugin, Theme |
+-----+ +-----+
| cPNK baserCMS Core | |
+-----+-----+
| CakePHP |
+-----+
| MySQL |
+-----+ +-----+ +-----+\ 
| cRED RED | cBLU BLU | |
+-----+-----+ +-----+
| cGRE GRE | cPNK PNK | |
+-----+-----+ +-----+
| cBLK BLK | cYEL YEL | |
\-----+-----+ / +-----+
-----
```

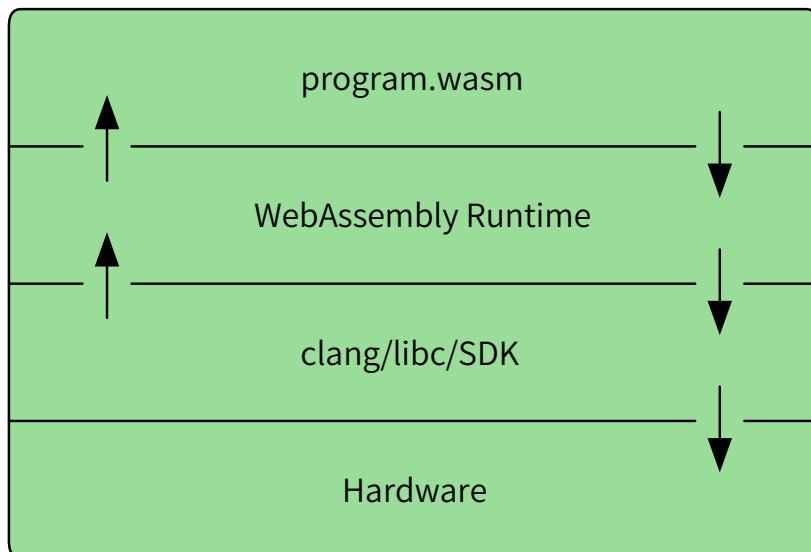


4. ダイアグラム記法

```
[ditaa, diag-ditaa-block-2, svg, shadows=true, separation=true, pdfwidth=70%, width=70%]
[caption=""]

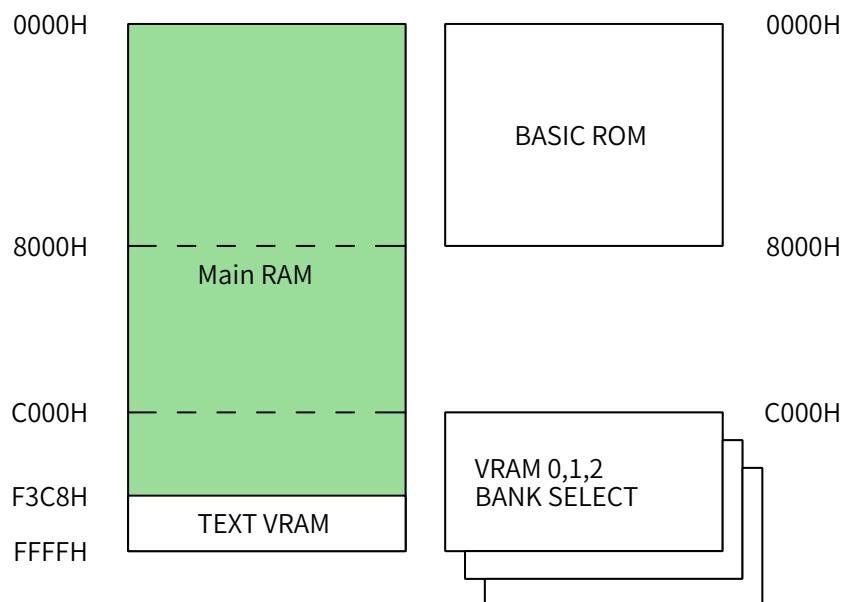
-----
/-----\ \
| | program.wasm |
| ^ | | |
+---+-----+---+
| | v | |
| WebAssembly Runtime |
| ^ | | |
+---+-----+---+
| | v | |
| clang/libc/SDK |
| | | |
+-----+---+
| | v | |
| cGRE Hardware |
| | | |
\-----/ |

-----
```



4.5. メモリーマップ

メモリーマップを dita 記法で記述します。



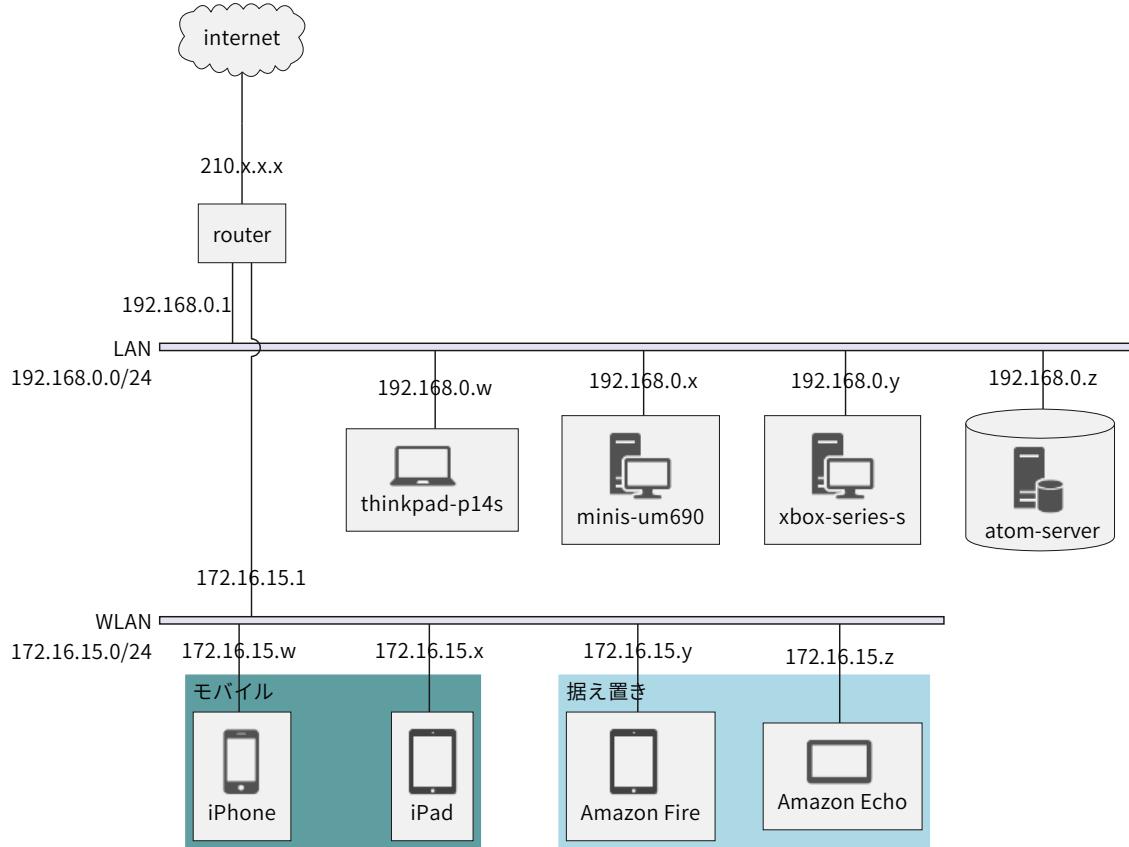
4.6. ネットワーク構成図

ネットワーク構成図を PlantUML の nwdiag で記述します。

diag-nwdiag-network1.puml - ソース より簡略化

```
[plantuml, diag-nwdiag-network1, svg]
-----
@startuml diag-network1
<style>
nwdiagDiagram {
    FontSize 14
    group {
        BackGroundColor cadetblue
    }
}
</style>

nwdiag {
    internet [shape = cloud];
    internet -- router;
    router [address = "210.x.x.x"];
    network LAN {
        address = "192.168.0.0/24"
        router [address = "192.168.0.1"];
        thinkpad-p14s [address = "192.168.0.w"];
        minis-um690 [address = "192.168.0.x"];
        xbox-series-s [address = "192.168.0.y"];
        atom-server [address = "192.168.0.z", shape = database"]
    }
    network WLAN {
        address = "172.16.15.0/24"
        router [address = "172.16.15.1"];
        group {
            description = モバイル
            iphone01 [address = "172.16.15.w"];
            ipad01 [address = "172.16.15.x"];
        }
        group {
            color = "#add8e6"
            description = 据え置き
            fire01 [address = "172.16.15.y"];
            echo01 [address = "172.16.15.z"];
        }
    }
}
@enduml
-----
```



4.7. システム構成図

簡単なシステム構成図を [PlantUML/C4 コンポーネント](#)で記述します。

```
[plantuml, diag-c4-component-sample1, svg]
[caption=""]
-----
@startuml diag-c4-component-sample1
!include <C4/C4_Container>
!include <office/users/users.puml>
!include <office/Servers/database_server>
!include <office/Servers/file_server>
!include <office/Servers/application_server>
!include <office/Concepts/service_application>
!include <office/Concepts/firewall>
!include <office/Devices/workstation>

skinparam linetype ortho
top to bottom direction
'left to right direction
HIDE_STEREOTYPE()

AddContainerTag("user", $bgColor="#5f9ea0", $shadowing="true")
AddContainerTag("linux", $sprite="application_server", $bgColor="#dcdcdc",
$fontColor="#000000", $borderColor="#696969", $sprite="file_server", $shadowing="true")

Person(user_person, "ウェブサイト\n閲覧者", $sprite="users", $tags="user")

System_Boundary(system_1, "Application") {
    Container_Boundary(bound_backend_1, "Backend") {
        ContainerDb(rel_db, "Database", "MySQL 8.0", $sprite="database_server", $tags="linux")
        Container(filesystem, "File System", "ext4", $tags="linux")
        Container(batch, "Batch Server", "Java, Spring Boot", $tags="linux")
    }
    Container_Boundary(bound_frontend_1, "Frontend") {
        Container(load_balancer, "Load balancer", "nginx", $sprite="application_server",
$tags="linux")
        Container_Boundary(bound_cluster_1, "Cluster") {
            Container(web_app_1, "Web Application", "Java, Spring Boot", $tags="linux")
            Container(web_app_2, "Web Application", "Java, Spring Boot", $tags="linux")
        }
    }
    Person(user_admin, "アプリケーション\n管理者", $sprite="workstation", $tags="user")
}

Lay_R(user_person, system_1)
Lay_D(user_admin, batch)
Lay_R(bound_backend_1, bound_frontend_1)

Lay_R(load_balancer, web_app_1)
Lay_D(web_app_1, web_app_2)
Lay_D(rel_db, filesystem)
Lay_R(rel_db, batch)

AddRelTag("rel_line", $textColor="black", $lineColor="#696969", $lineStyle=BoldLine())
```

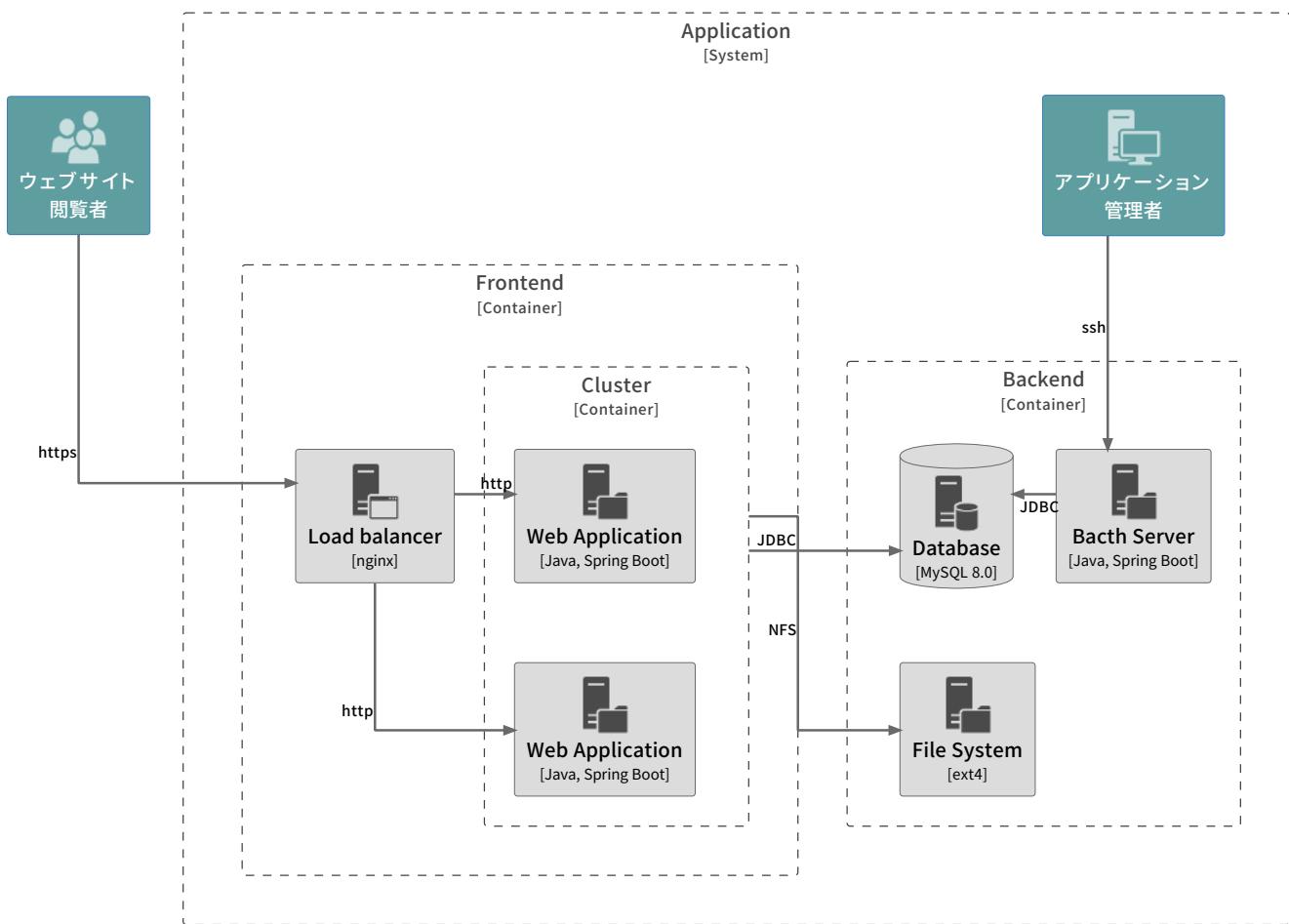
```

Rel_R(user_person, load_balancer, "https", $tags="rel_line")
Rel_R(load_balancer, web_app_1, "http", $tags="rel_line")
Rel_R(load_balancer, web_app_2, "http", $tags="rel_line")

Rel_R(bound_cluster_1, rel_db, "JDBC", $tags="rel_line")
Rel_R(bound_cluster_1, filesystem, "NFS", $tags="rel_line")

Rel_R(batch, rel_db, "JDBC", $tags="rel_line")
Rel_R(user_admmin, batch, "ssh", $tags="rel_line")

@enduml
----
```



現在のところ \$shadowing="true" 指定のドロップシャドーは HTML 版のみ反映します。

4.8. タイミングチャート

デジタル回路などのタイミングチャートを PlantUML で記述します。

diag-timing-sample1.puml - ソース より抜粋

```
[plantuml, diag-timing-sample1, svg, pdfwidth=90%, width=80%]
-----
@startuml diag-timing-sample1
scale 40 as 150 pixels
clock "Clock" as clk with period 1
binary "CS" as CS
binary "WR" as WR
binary "RD" as RD
binary "A0" as A0
binary "A1" as A1
binary "IC" as IC
concise "DataBus" as DB

@0 as :start
@20 as :set_data_bus_1
@30 as :set_addr_1
@40 as :write_start_1
@60 as :write_end_1

@:start
IC is high
CS is high
WR is high
RD is high
A0 is low
A1 is low

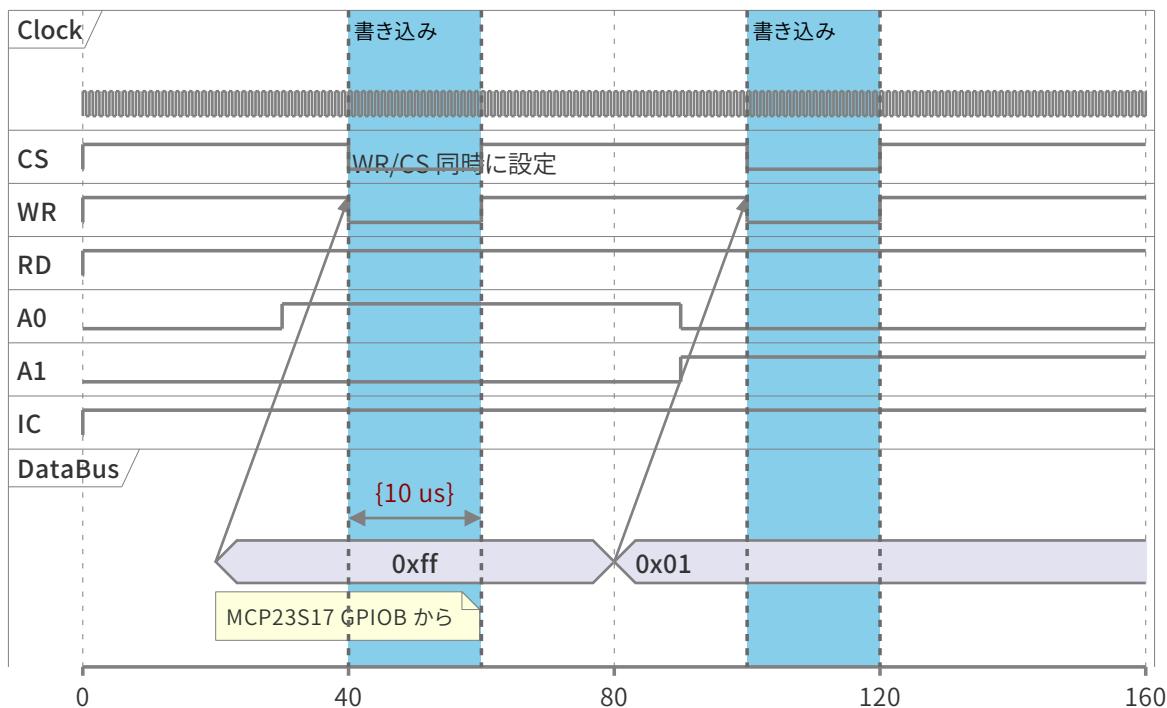
@:set_data_bus_1
DB is "0xff"
DB -> WR@+20
note bottom of DB : MCP23S17 GPIOB から

@:set_addr_1
A0 is high
A1 is low

@:write_start_1
CS is low : WR/CS 同時に設定
WR is low
DB@40 <-> @60 : {10 us}

@:write_end_1
CS is high
WR is high

highlight 40 to 60 #SkyBlue;line:DimGrey : 書き込み
@enduml
-----
```



Open Iconic アイコン

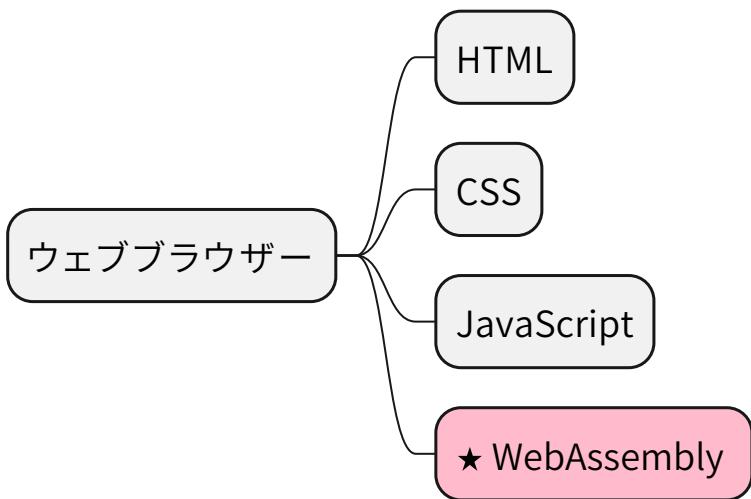
PlantUML 内で <&アイコン名> 形式で使える Open Iconic アイコンは次の通りです。アイコン名は HTML 版の場合 ウェブブラウザ › 右クリック › 画像を開く するとコピーアンドペーストで取得できます。

List Open Iconic	▲ bell	◆ cloud	≡ excerpt	≡ justify-right	♪ musical-note	★ star
Credit to	* bluetooth	▲ cloudy	≡ expand-down	♪ key	◐ paperclip	* sun
https://useiconic.com/open	■ bold	◦ code	≡ expand-left	□ laptop	◑ pencil	□ tablet
	◆ bolt	◎ cog	≡ expand-right	■ layers	● people	❖ tag
→ account-login	■ book	≡ collapse-down	≡ expand-up	◊ lightbulb	▲ person	» tags
→ account-logout	■ bookmark	≡ collapse-left	≡ external-link	○ link-broken	□ phone	◎ target
↷ action-redo	■ box	≡ collapse-right	○ eye	○ link-intact	◑ pie-chart	☒ task
↶ action-undo	■ briefcase	≡ collapse-up	○ eyedropper	■ list-rich	† pin	▣ terminal
≡ align-center	£ british-pound	■ command	● file	■ list	● play-circle	Ｔ text
≡ align-left	□ browser	■ comment-square	● fire	↗ location	+ plus	▼ thumb-down
≡ align-right	✓ brush	○ compass	● flag	● lock-locked	○ power-standby	▲ thumb-up
⌚ aperture	* bug	● contrast	● flash	● lock-unlocked	◐ print	◐ timer
↓ arrow-bottom	■ bullhorn	≡ copywriting	■ folder	○ loop-circular	◑ project	▬ transfer
● arrow-circle-bottom	■ calculator	■ credit-card	● fork	■ loop-square	● pulse	■ trash
● arrow-circle-left	■ calendar	■ crop	■ fullscreen-enter	≡ loop	■ puzzle-piece	underline
● arrow-circle-right	■ camera-slr	○ dashboard	■ fullscreen-exit	○ magnifying-glass	? question-mark	■ vertical-align-bottom
● arrow-circle-top	▼ caret-bottom	■ data-transfer-download	○ globe	○ map-marker	● rain	▣ vertical-align-center
← arrow-left	◀ caret-left	■ data-transfer-upload	○ graph	■ map	× random	■ vertical-align-top
→ arrow-right	▶ caret-right	● delete	■ grid-four-up	■ media-pause	○ reload	■ video
↓ arrow-thick-bottom	▲ caret-top	● dial	■ grid-three-up	▶ media-play	● resize-both	● volume-high
← arrow-thick-left	▼ caret	■ document	■ grid-two-up	● media-record	↑ resize-height	● volume-low
→ arrow-thick-right	■ chat	§ dollar	■ hard-drive	● media-skip-backward	↑ resize-width	● volume-off
↑ arrow-thick-top	✓ check	” double-quote-sans-left	■ header	● media-skip-forward	● rss-alt	▲ warning
↑ arrow-top	▼ chevron-bottom	” double-quote-sans-right	○ headphones	● media-step-backward	● rss	❖ wifi
◐ audio-spectrum	◀ chevron-left	” double-quote-serif-left	● heart	● media-step-forward	■ script	◑ wrench
◑ audio	▶ chevron-right	” double-quote-serif-right	● home	■ media-stop	◐ share	✖ x
● badge	▲ chevron-top	● droplet	■ image	● medical-cross	● shield	¥ yen
◐ ban	● circle-check	▲ eject	■ inbox	≡ menu	● signal	◐ zoom-in
◑ bar-chart	● circle-x	● elevator	∞ infinity	● microphone	● sort-ascending	◐ zoom-out
◑ basket	■ clipboard	○ ellipsis	○ info	— minus	● signpost	● sort-descending
□ battery-empty	○ clock	■ envelope-closed	■ italic	■ monitor	● spreadsheet	
■ battery-full	♦ cloud-download	● envelope-open	≡ justify-center	● moon		
■ beaker	♦ cloud-upload	€ euro	≡ justify-left	+ move		

4.9. マインドマップ

PlantUML のマインドマップ記法を使い木構造を表現します。

```
[plantuml, diag-mindmap-sample1, svg]
[caption="""]
-----
@startmindmap diag-mindmap-sample1
<style>
mindmapDiagram {
    FontSize 18
    .rose {
        BackgroundColor #ffbbcc
    }
}
</style>
* ウェブブラウザー
** HTML
** CSS
** JavaScript
** <&star> WebAssembly <<rose>>
@endjson
-----
```



奥付

著者紹介

田中 広将 (たなか ひろまさ)

ソフトウェア開発者。近年はハードウェア工作にも注目し、オープンソースハードウェアの公開なども行っている。座右の銘は「論よりラン」。

Asciidoctor と Gradle でつくる執筆環境

2023年9月14日 v4.1.0 発行

著者 田中 広将 (ひろまさ - h1romas4)

カバーデザイン Fujix

発行 <https://github.com/h1romas4/asciidocor-gradle-template>

ライセンス MIT License

