

Asciiidoctor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ガイドです。実行環境は macOS、Linux、Windows の各 OS に対応しています。

この文書の手順から次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 ビルドの実行

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 拡張の設定

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軽量なマークアップ言語です。他の軽量テキストマークアップ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を用いた執筆とパブリッシング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、本文書ではバージョン管理システムの Git や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フォローしています。

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。

AsciiDoc と AsciiDoctor について

AsciiDoc は人が読み書きしやすいプレーンテキスト文書から、HTML や PDF をはじめとした様々な形式に出力が行える軽量のマークアップ言語です。通常のテキストファイルを使った表現と似ていることから習得がしやすく、また出版向きの表現力も兼ね備えています。

About AsciiDoc

AsciiDoc is a lightweight and semantic markup language primarily designed for writing technical documentation. The language can be used to produce a variety of presentation-rich output formats, all from content encoded in a concise, human-readable, plain text format.

— AsciiDoc Language Documentation

AsciiDoctor は Ruby でかかれた AsciiDoc の実装です。構文木解析や各種出版フォーマットへの変換を実現します。

<https://docs.asciidoctor.org/asciidoctor/latest/>

What is AsciiDoctor?

AsciiDoctor is a fast, open source, text processor for parsing AsciiDoc into a document model, then converting it to output formats such as HTML 5, DocBook 5, man(ual) pages, PDF, and EPUB 3. AsciiDoctor is written in the Ruby programming language.

— AsciiDoctor Documentation

同プロジェクトにおいて、ウェブブラウザ上で AsciiDoc の動作を実現する AsciiDoctor.js や、PDF 出力を行う AsciiDoctor PDF が提供されています。また、AsciiDoctorJ は、Ruby 実装である AsciiDoctor を Java でかかれた Ruby 実行環境である JRuby を使って Java VM 上で動作させるプロダクトです。

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`、ダイアグラムの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/asciidoctor-gradle-template>

Asciidoctor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

- Asciidoctor - MIT License - <https://asciidoctor.org/>
- Asciidoctorj - Apache License 2.0 - <https://github.com/asciidoctor/asciidoctorj>
- Asciidoctor.js - MIT License - <https://asciidoctor.org/docs/asciidoctor.js/>
- Asciidoctor PDF - MIT License - <https://asciidoctor.org/docs/asciidoctor-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>
- Draw.io - Apache License 2.0 - <https://github.com/jgraph/drawio>
- vscode-drawio - GPL-3.0 license - <https://github.com/hediet/vscode-drawio>

Java Development Kit & Build Tool

- Eclipse Temurin - Multi(Apache License 2.0/GPLv2+CE/GPLv2+Assembly) - <https://adoptium.net/about/>
- 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>
- Code Spell Checker - GPL - <https://github.com/streetsidesoftware/vscode-spell-checker>

Guide

- asciidoctor-pdfでかつこいい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
AsciiDoc と AsciiDoctor について	2
ライセンス	3
謝辞	3
1. AsciiDoc 文書変換用スクリプトを使う準備	7
1.1. Java 実行環境の導入 (macOS / Linux)	8
1.2. Java 実行環境の導入 (Windows)	10
2. AsciiDoc から HTML/PDF 文書を作成する	14
2.1. サンプル文書の変換を試す	14
2.2. 変換後の文書	17
2.3. テキストエディタで AsciiDoc 文書を編集する	19
2.4. 文書のファイル構成	29
2.5. 執筆の開始	31
2.6. 執筆のイテレーション	31
2.7. 文書の Git 管理	33
2.8. クラウド環境による執筆	39
2.9. クラウド環境でのファイル保存と執筆の再開	43
2.10. GitHub Actions によるビルド	46
3. AsciiDoc 記法	50
3.1. ドキュメントヘッダーとセクションタイトル	50
3.2. 段落	51
3.3. 文章中のテキストフォーマット	51
3.4. 文書中の相互参照	52
3.5. 外部リンク	52
3.6. 引用	53
3.7. 画像	53
3.8. ソースコードシンタックスハイライト	54
3.9. メニュー・ボタン	55
3.10. ファイルインクルード	55
3.11. 表形式	55
3.12. リスト	57
3.13. 脚注	58
3.14. 改ページ・ブレイク表現	59
3.15. ラベル文言	59
3.16. コラム表現	60
3.17. コメント行	60
4. ダイアグラム記法	61
4.1. シーケンス図	62
4.2. フォルダ・ファイルツリー	64
4.3. 数式	65
4.4. ブロック図	66
4.5. メモリーマップ	68
4.6. ネットワーク構成図	69
4.7. システム構成図	71
4.8. タイミングチャート	73
4.9. マインドマップ	75
5. Gradle ビルドの詳細	77
5.1. Gradle タスクの実行方法	77
5.2. tasks タスク	78

5.3. docs タスク	78
5.4. checkPdfInfo タスク	82
5.5. checkSyncImage タスク	83
奥付	85

1. AsciiDoc 文書変換用スクリプトを使う準備

本文書で用いる 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環境の準備は整っています。

手順. 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)
```

手順. 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、ビルド時にいくつか内部処理のワーニングが出力されますが Java 17/21 の各 LTS バージョンでも期待通り動作することを確認しています。

Graphviz の導入

macOS 及び Linux で、後述するダイアグラム記法の全てを変換するには Graphviz 画像ライブラリの導入が必要です。この文書のサンプルソースでも依存し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 の場合は内部で処理されるため、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 上に 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 ❶
$ source "$HOME/.sdkman/bin/sdkman-init.sh" ❷
$ sdk list java ❸
```

```
=====
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 ❹
```

- ❶ SDKMAN を導入します。
- ❷ SDKMAN を環境に設定します。
- ❸ 導入できる Java のバージョンを一覧します。
- ❹ 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、この後で実行する Gradle ビルドツールが 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 ❶  
export JAVA_HOME=~/.sdkman/candidates/java/current ❷  
$ source ~/.bashrc ❸
```

- ❶ エディタで `.bashrc` や `.zshrc` を開きます。
- ❷ 本ラインをファイルの最下部に追加し保存終了します。
- ❸ 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。



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。

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 ❶  
$ sdk default java 11.0.20.1-tem ❷  
$ sdk default java 17.0.4.1-tem ❸
```

- ❶ Java 11 を導入
- ❷ Java 11 をデフォルトに設定
- ❸ Java 17 をデフォルトに戻す

1.2. Java 実行環境の導入(Windows)

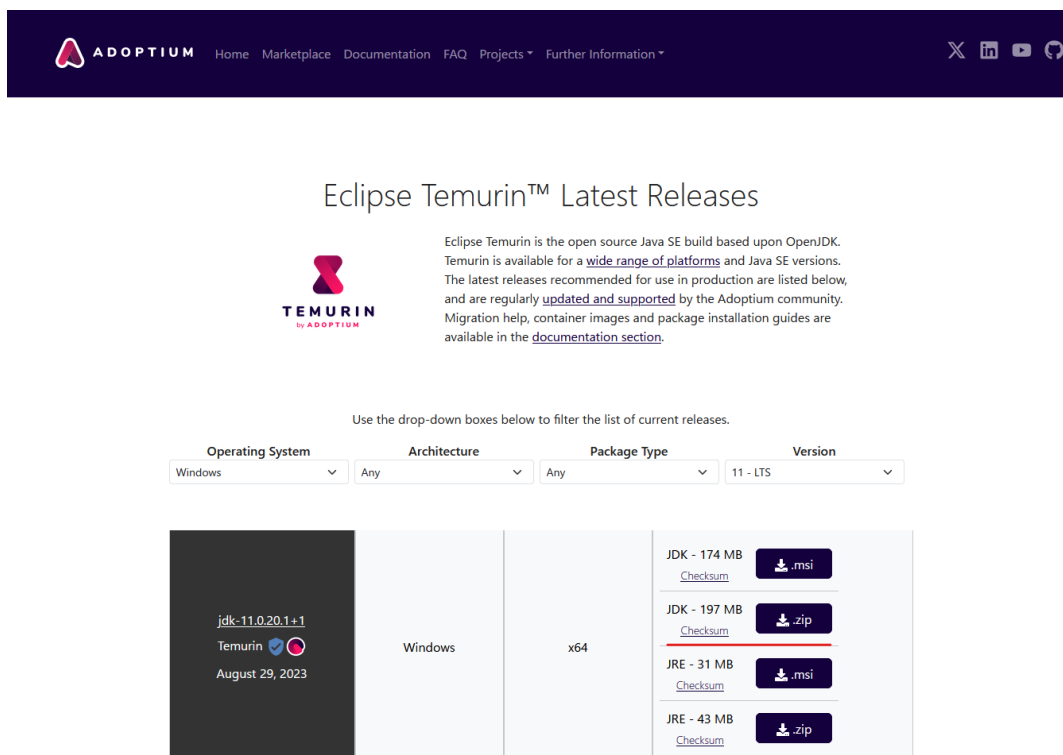
もし Windows 上に Java 実行環境がなければ Eclipse Temurin プロジェクトが提供する OpenJDK のバイナリを導入すると良いでしょう。

<https://adoptium.net/temurin/releases/?os=windows&version=11>

Eclipse Temurin is the open source Java SE build based upon OpenJDK.

— AdoptOpenJDK

サイトにブラウザでアクセスし、jdk-11 の最新版の中から JDK - zip ファイルをダウンロードしてください。



Eclipse Temurin™ Latest Releases

Eclipse Temurin is the open source Java SE build based upon OpenJDK. Temurin is available for a [wide range of platforms](#) and Java SE versions. The latest releases recommended for use in production are listed below, and are regularly [updated](#) and [supported](#) by the Adoptium community. Migration help, container images and package installation guides are available in the [documentation section](#).

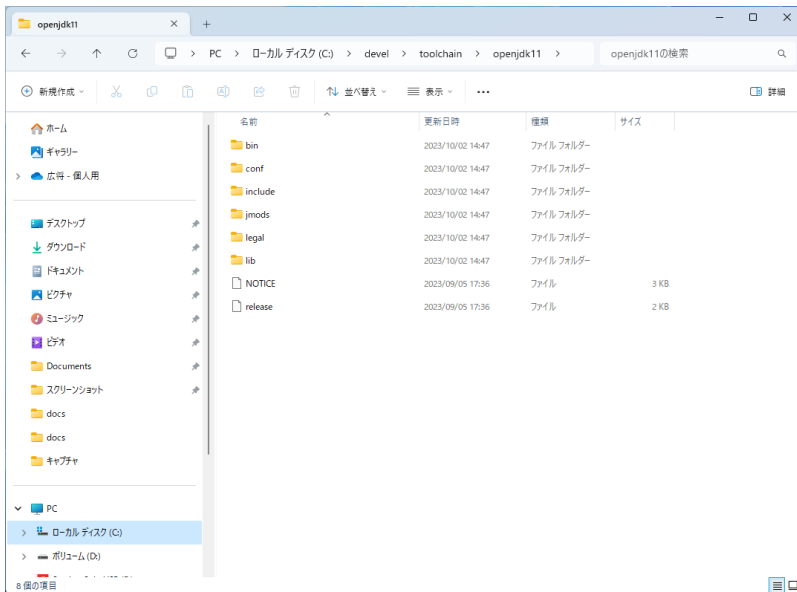
Use the drop-down boxes below to filter the list of current releases.

Operating System	Architecture	Package Type	Version
Windows	Any	Any	11 - LTS

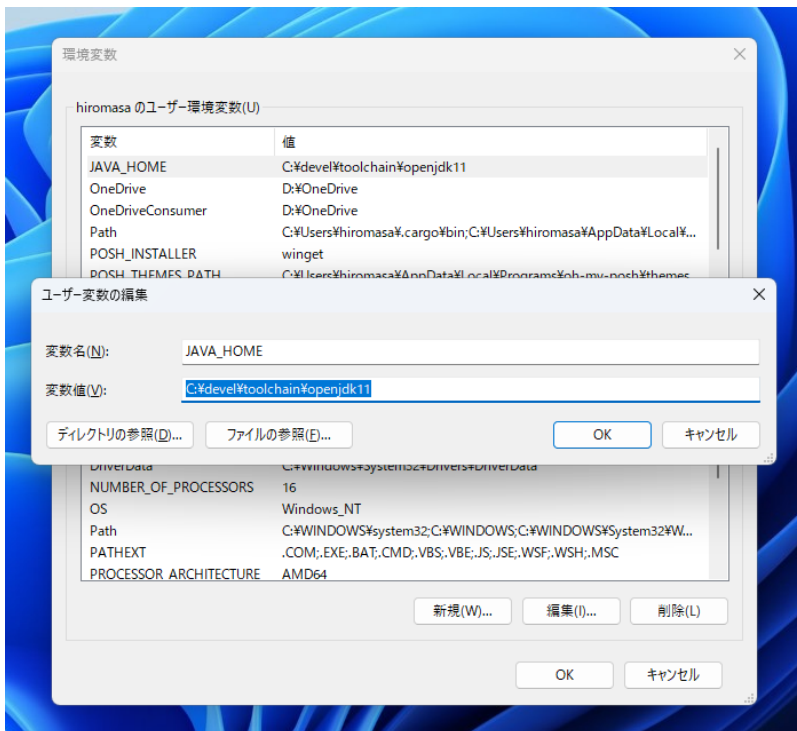
Release Information	Architecture	Package Type
jdk-11.0.20.1+1 Temurin August 29, 2023	Windows	x64

JDK - 174 MB Checksum	.msi
JDK - 197 MB Checksum	.zip
JRE - 31 MB Checksum	.msi
JRE - 43 MB Checksum	.zip

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



また、この後で実行する Gradle ビルドツールが 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** > **システム** > **システムの詳細設定** > **環境変数** 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\toolchain\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 24.04 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 24.04 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。

手順. WSL2 Ubuntu 24.04 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 ❶
$ sudo update-locale LANG=ja_JP.UTF8 ❷
$ echo 'export LANG=ja_JP.UTF-8' >> ~/.bashrc ❸
$ fc-cache -f ❹
```

- ❶ 日本語の言語設定とデフォルトフォント関連の導入。
- ❷ 日本語ロケールを追加。
- ❸ 起動シェル環境を `ja_JP.UTF-8` に設定。
- ❹ 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。

手順. WSL2 Ubuntu 24.04 LTS への Java の導入

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

- ❶ SDKMAN を導入。
- ❷ SDKMAN を環境に設定。
- ❸ Java 11 系を指定して Java を導入。
- ❹ `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。

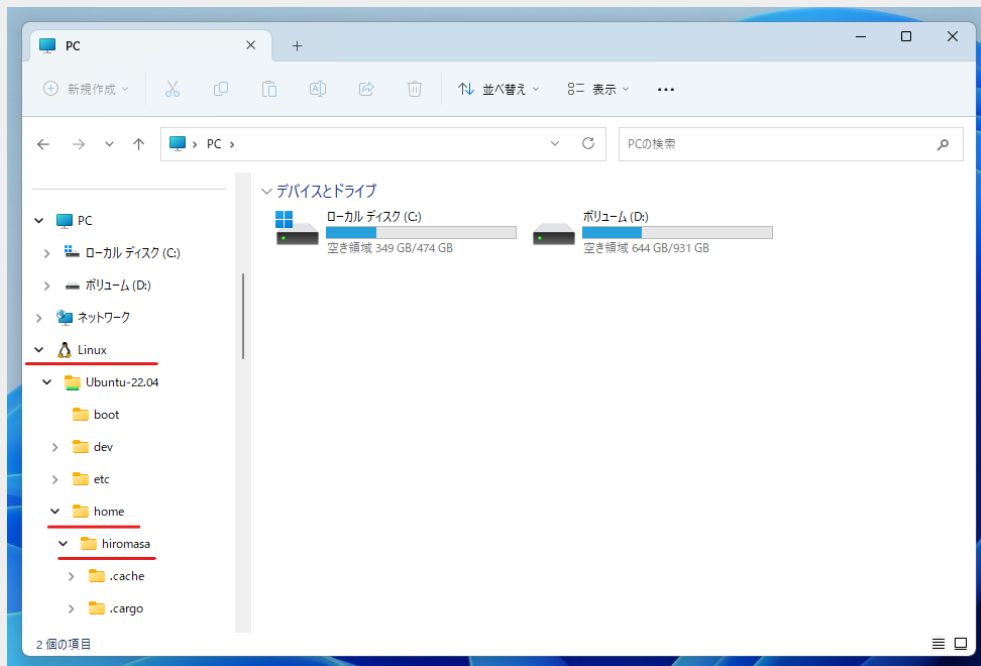


次項以降 WSL2 Ubuntu 環境を使う場合の手順は、特別な記載がある場合を除き Linux 項を参照して操作してください。

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。

文書変換スクリプトの実行には、前項で導入した Java 上で動作する **Gradle** ビルドツールが使われ、リポジトリに含まれる `gradlew` シェルスクリプトを起点として各種設定が自動的に行われ、AsciiDoc 文書から HTML/PDF 文書への変換が実行されます。本文書ではこれを、文書のビルドもしくは単純にビルドと呼ぶこととします。



`gradlew` コマンドは `gradlew [タスク名]` として実行したい機能呼び分けられます。`docs` 部分はこの機能を用い、ビルドスクリプト内で文書ビルドのタスクと定義しています。タスク名称や操作の内容は、リポジトリ中の `build.gradle` ファイル中に含まれます。

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 AsciiDoc 文書を HTML/PDF 変換するためにビルドを実行します。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` コマンドでリポジトリファイルの取得を行い、`.zip` の展開とビルドの実行を行います。

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/hlromas4/asciidoc-gradle-template/archive/main.zip ❶
$ unzip main.zip ❷
$ cd asciidoc-gradle-template-main ❸
$ ./gradlew docs ❹
BUILD SUCCESSFUL in 1m 13s 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 1分程度で完了します。
- ❺ `BUILD SUCCESSFUL` が出力されれば文書変換が正常終了しています。

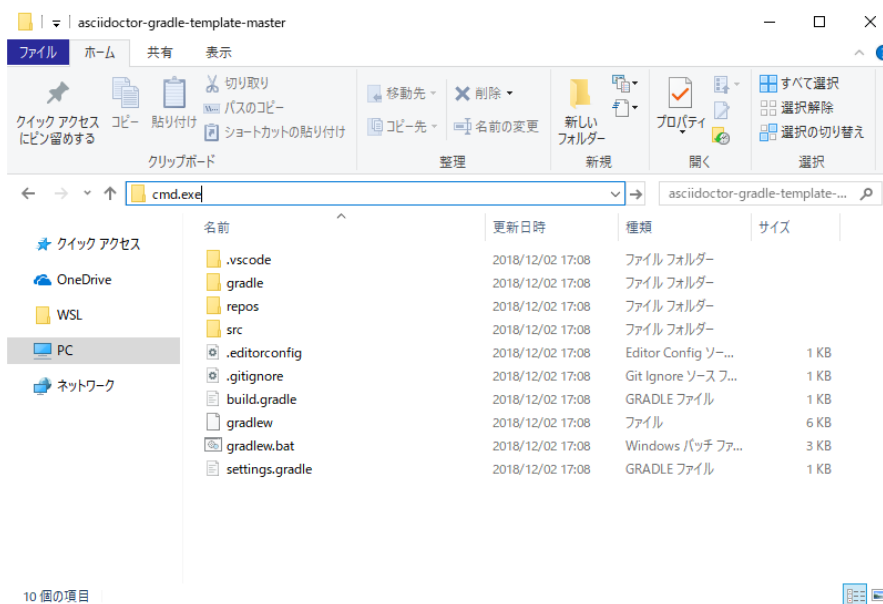
2.1.2. サンプル文書の変換(Windows)

ウェブブラウザで URL にアクセスしてリポジトリファイルの取得を行い、エクスプローラで `.zip` 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/asciidoctor-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` もしくは `powershell.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 1分程度で完了します。

```
> .\gradlew docs
BUILD SUCCESSFUL in 1m 10s ❶
2 actionable tasks: 2 executed
```

- ❶ BUILD SUCCESSFUL が出力されれば文書変換が正常終了しています。

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、プロジェクトフォルダの `docs` 配下に 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

ビルド結果の出力先となる HTML/PDF 文書フォルダ (docs) フォルダの構成は次のとおりです。

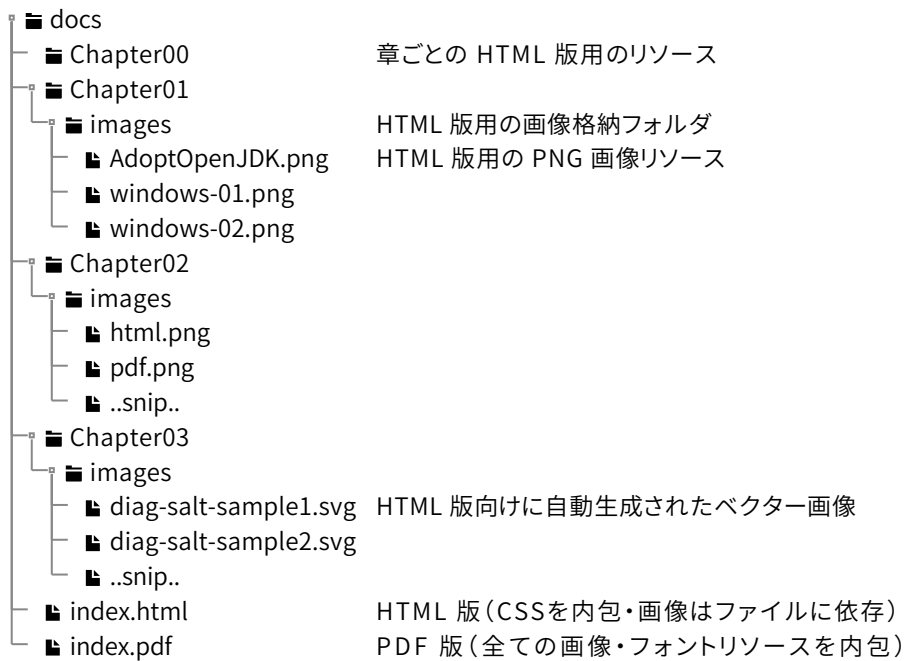


図. HTML/PDF 文書フォルダ (docs)

ビルド対象の文書となるソースフォルダ (src/docs/asciidoc) の構成は次のとおりです。

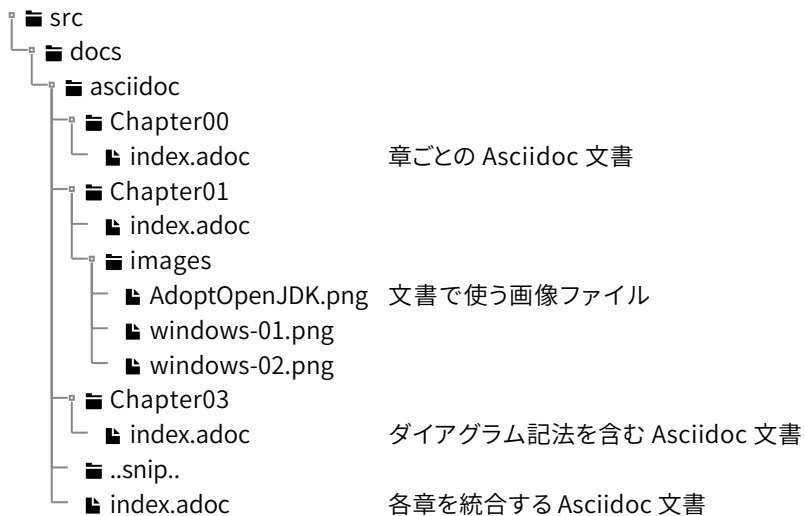


図. ビルド対象のソースフォルダ (src/docs/asciidoc)

ビルドの基本的な動作は次のようになります。

- HTML/PDF 文書フォルダ (docs) 内のファイルを全て削除する。
- ソースフォルダ (src/docs/asciidoc) に配置された AsciiDoc 文書と画像などのリソースを変換処理し、HTML/PDF 文書フォルダに格納する。
 - AsciiDoc 文書に後述するダイアログ記法による図表が存在していた場合はベクター画像生成する。
 - HTML 版の文書は docs/index.html として生成。docs/Chapter*/images フォルダに配置した画像などのリソースをリンクする。

- PDF 版の文書は docs/index.pdf として生成。フォントや画像リソースなど全てを PDF ファイル内に格納する。

以上からビルドした文書の配布は次のようになります。

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



HTML/PDF 文書フォルダ (docs) はビルドした文書を配置する専用フォルダです。ソースフォルダと同期するためビルド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。

なお、この後解説する Visual Studio Code テキストエディタでは、プロジェクト設定で docs フォルダをリードオンリーに指定し、誤ってファイルを追加・修正できないように制御してあります。

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。

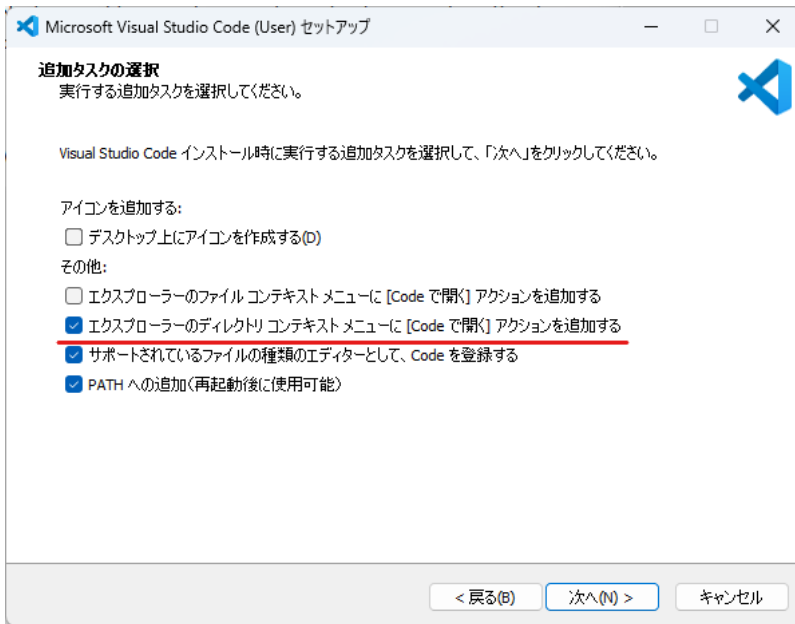
次のサイトからお使いの OS に合わせた VS Code を導入してください。

<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。この拡張に処理により、VS Code のバックエンド処理が Ubuntu 側で、フロントエンド画面が Windows 側で動作できるようになります。

<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 バックエンドの初期セットアップが動作し、その後 Windows 上で VS Code フロントエンド画面が立ち上がります。

手順. WSL2 Ubuntu 24.04 への VS Code サーバの導入

```
$ code ❶
```

- ❶ 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。

手順. WSL2 Ubuntu 24.04 からの VS Code 起動

```
$ cd asciidoctor-gradle-template ❶  
$ code . ❷
```

- ❶ 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。
- ❷ `.` 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 拡張の詳細は次のドキュメントから得られます。

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

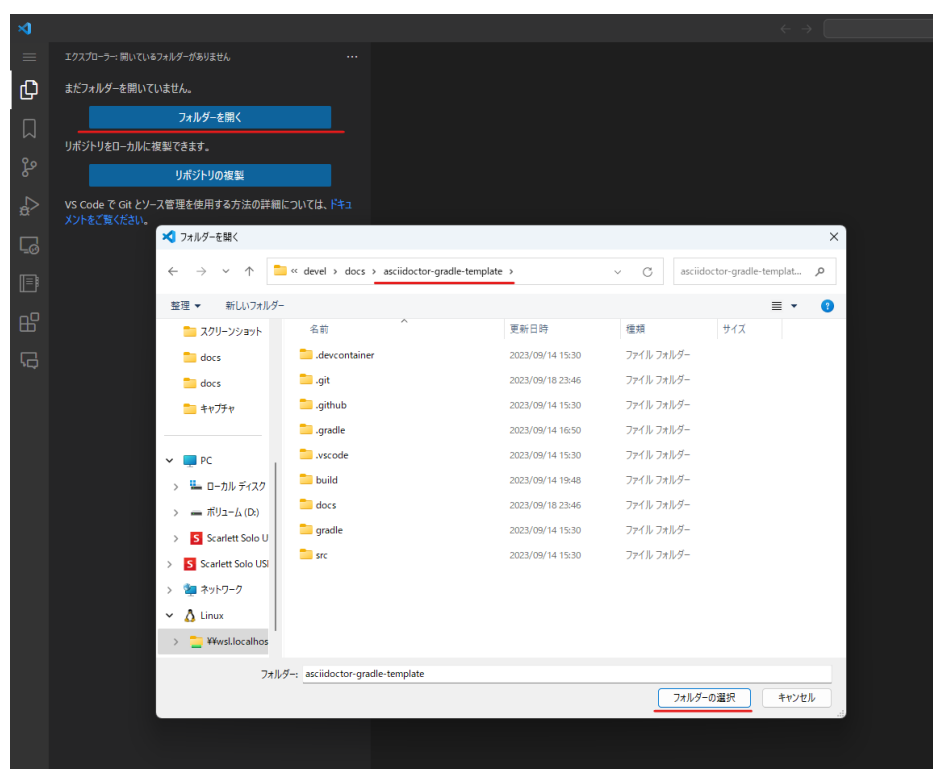
The Visual Studio Code WSL extension lets you use the Windows Subsystem for Linux (WSL) as your full-time development environment right from VS Code. You can develop in a Linux-based environment, use Linux-specific toolchains and utilities, and run and debug your Linux-based applications all from the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



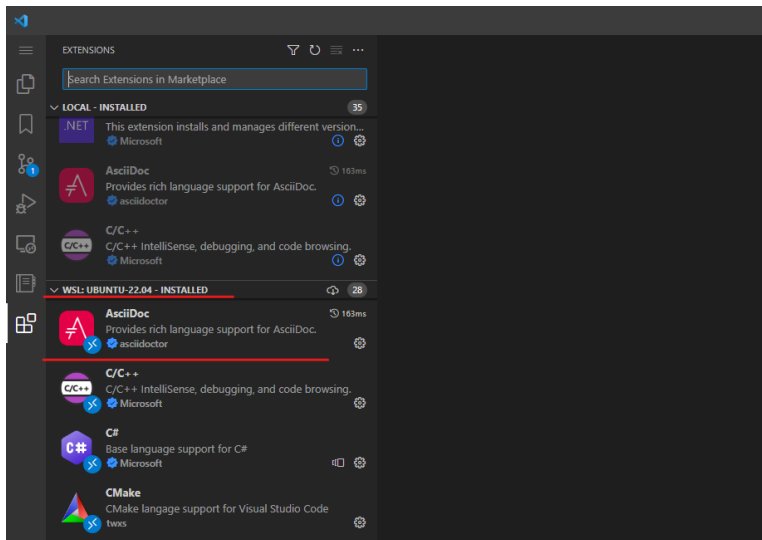
プロジェクトフォルダのオープンは、エクスプローラなどのシェルからフォルダを右クリックして表示されるコンテキストメニューの **[Code で開く]** 操作や、ターミナル画面からカレントディレクトリを `cd` コマンドで移動し `code .` と入力することでも可能です。

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。

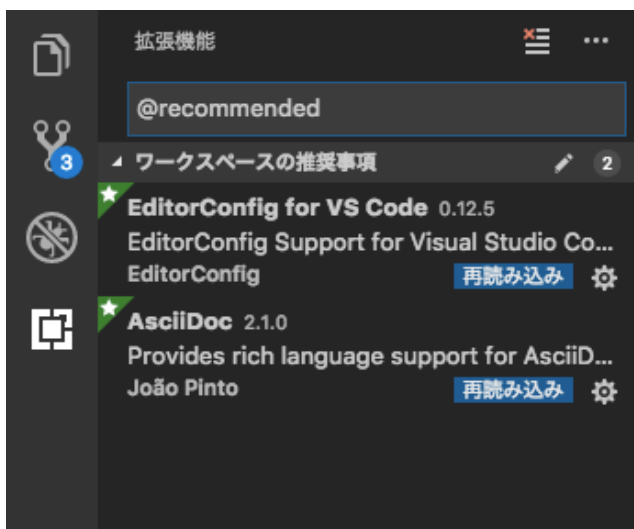
1. このワークスペースには拡張機能の推奨事項があります。▶ **すべてインストール** ボタンをクリックします。



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。



VS Code のリアルタイムプレビュー設定は `.vscode/settings.json` ファイルで、導入推奨拡張は `.vscode/extensions.json` ファイルで定義されています。

これらのファイルを修正することで、例えば拡張に日本語校正やスペルチェックを追加し設定を定義するなど、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。



```
index.adoc x
src > docs > asciidoc > Chapter01 > index.adoc > AsciiDoc 文書変換用スクリプトを使う準備
1 include::../attribute.adoc[]
2
3 == AsciiDoc 文書変換用スクリプトを使う準備
4
5 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。
6
7 お使いのコンピュータ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環境の準備は整ってい
  ます。
8
9 [source]
10 [caption=""]
11 macOS / Linux の場合
```

画面. AsciiDoc 拡張 Open Preview to the Side



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。



```
index.adoc .../Chapter03 ... index.adoc .../Chapter01 ... Preview index.adoc x
4 == AsciiDoc から HTML/PDF 文書を作成する
5
6 == サンプル文書の変換を試す
7
8 環境の準備ができましたので AsciiDoc 文書を HTML/PDF に変換してみます。
9
10 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、リポジトリに
  は HTML/PDF 変換に使うファイル一式と、文書サンプルとして "この文書" の
  AsciiDoc ファイルが置かれています。まずはサンプル文書が正しく変換できるか
  を試してみましょう。
11
12 macOS / Linux の場合は次のようにします。
13
14 [source]
15 [caption="手順. "]
16 .PDF 変換ビルドスクリプトの取得と実行
17 ----
18 $ curl -L -O
  https://github.com/hlromas4/asciidoc-gradle-template/archive/master.
  zip # <1>
19 $ unzip master.zip # <2>
20 $ cd asciidoc-gradle-template-master # <3>
21 $ ./gradlew docs # <4>
22 BUILD SUCCESSFUL in 19s <5>
23 2 actionable tasks: 2 executed
24 ----
25
26 <1> リポジトリのファイルをダウンロードします。
27 <2> ダウンロードした .zip ファイルを展開します。
28 <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。
29 <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。今回は数秒で完了します。
30 <5> ``BUILD SUCCESSFUL`` が出力されればビルド成功です。
```

画面. AsciiDoc のリアルタイムプレビュー

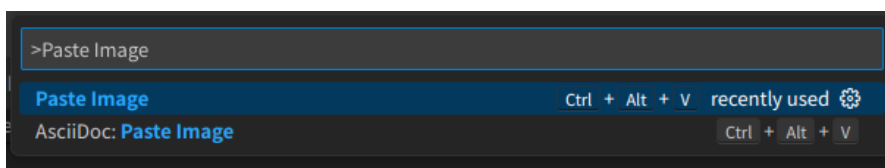
.adoc ファイルには本文書の内容がそのまま AsciiDoc 文書形式で書かれているのが分かります。いくつか開いて試し入力などをしてみると、AsciiDoc 文書のリアルタイムプレビューを含め執筆の雰囲気が掴めるはずです。

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"
}
```

なお、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ことから、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

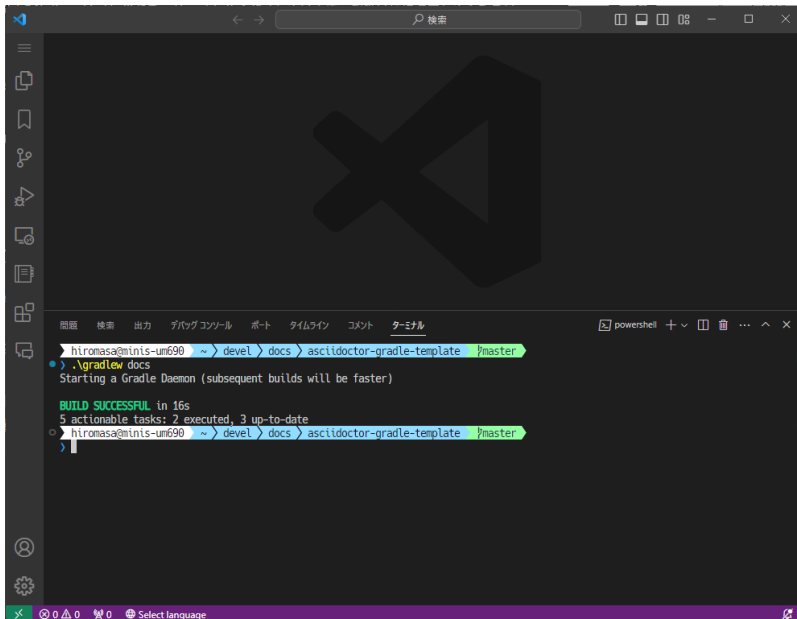
```
#!/bin/sh
# change current directory
cd `dirname $0`
# set dist path
DISTPATH=//wsl.localhost/Ubuntu-22.04 ❶
# call Windows PowerShell
powershell.exe -ExecutionPolicy Bypass -File pc.ps1 ${DISTPATH}$1
```

❶ 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動作させています。ファイルをリードオンリーに指定するなどして、取扱いには注意してください。

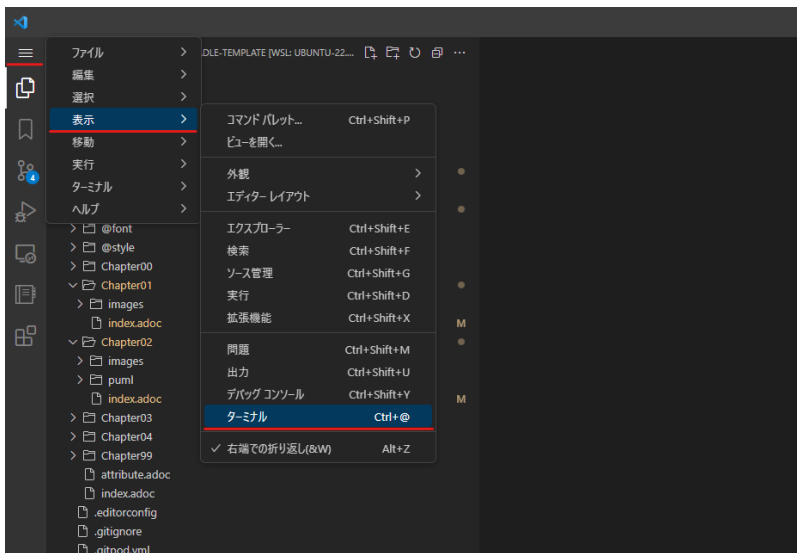
2.3.5. 統合ターミナルの活用

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、エディターを離れることなく文書のビルドが可能です。



画面. 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。



画面. VS Code 統合ターミナルの起動

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。再度、同キーを押下することでクローズするトグル動作となっており、画面を広く使いたい執筆中に活用できます。



統合ターミナル内で起動する一般的なシェル環境では、**[↑]** 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

本文書では 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。

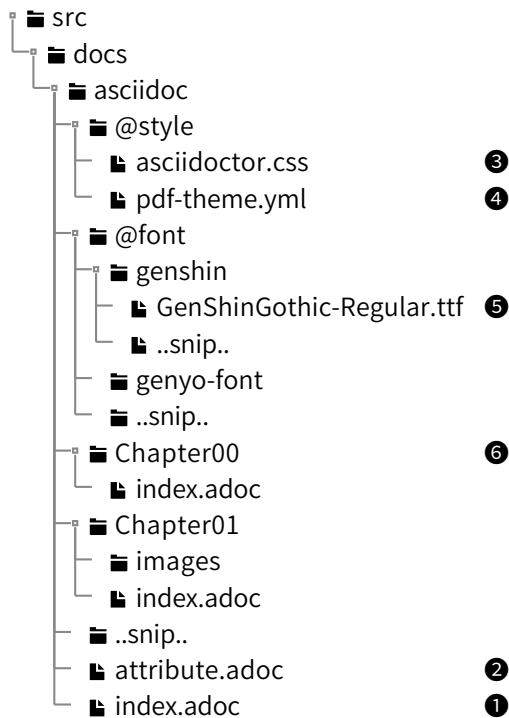


画像. IntelliJ IDEA の AsciiDoc Plugin で文書編集している例

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) のファイル構成の詳細を示します。



```

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

```

- 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。各子文書のフォルダ名は `Chapter{number}` とします。

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く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値を定義できます。画像挿入時の「図。」など標準で付与される文字列がありますので、必要に応じて修正します。

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のドキュメントから参照できます。

<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・行送り幅などの設定がありますので、必要に応じて修正します。

PDF テーマの詳細は次のドキュメントから参照できます。

<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われば、次のような流れで新しい執筆を開始できるでしょう。

1. サンプル文書の `src/docs/asciidoc/Chapter02` 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/Chapter00/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/Chapter01/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

ここまでの手順で執筆環境が整った後、毎日の執筆の進め方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/asciidocs` 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/asciidocs/index.adoc` に Chapter を追記し、`src/docs/asciidocs/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/asciidocs/Chapter*/images` フォルダに追加する。画像の追加は VS Code の Paste Image 拡張を用いることで、クリップボード経由で自動的行えます。
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` や `docs/index.pdf` で推敲する。
 - `./gradlew docs` の入力には 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

お使いのコンピュータに 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。

手順. macOS(ターミナルエミュレータで実行)

```
brew install git
```

手順. Linux(ターミナルエミュレータで実行)

```
# Ubuntu  
sudo apt install git
```

手順. Windows(PowerShell ウィンドウで実行)

```
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。

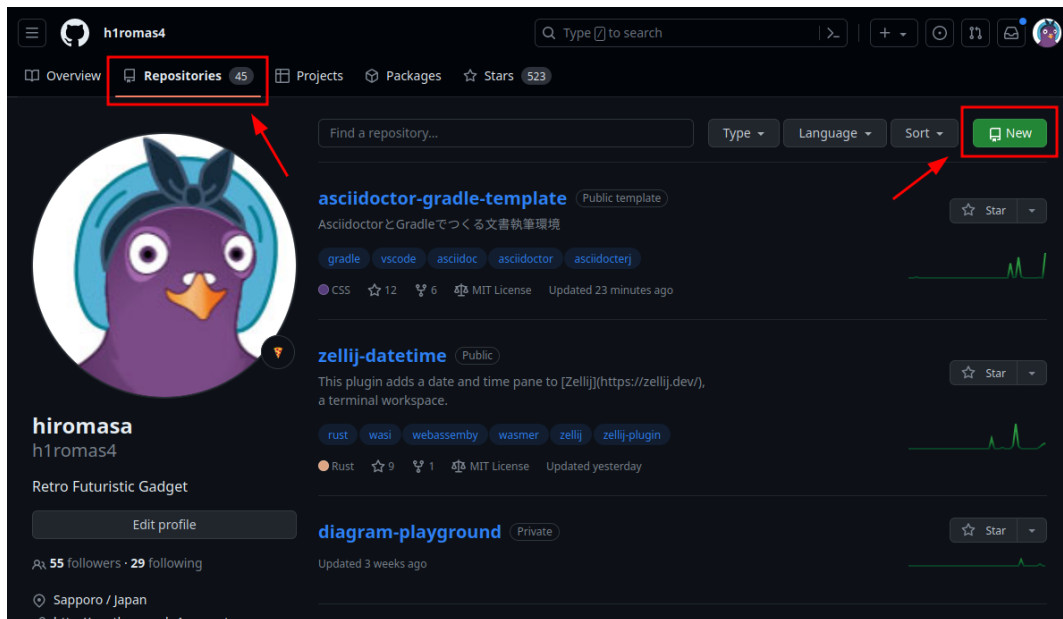
<https://docs.github.com/ja/authentication/connecting-to-github-with-ssh/adding-a-new-ssh-key-to-your-github-account>

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。

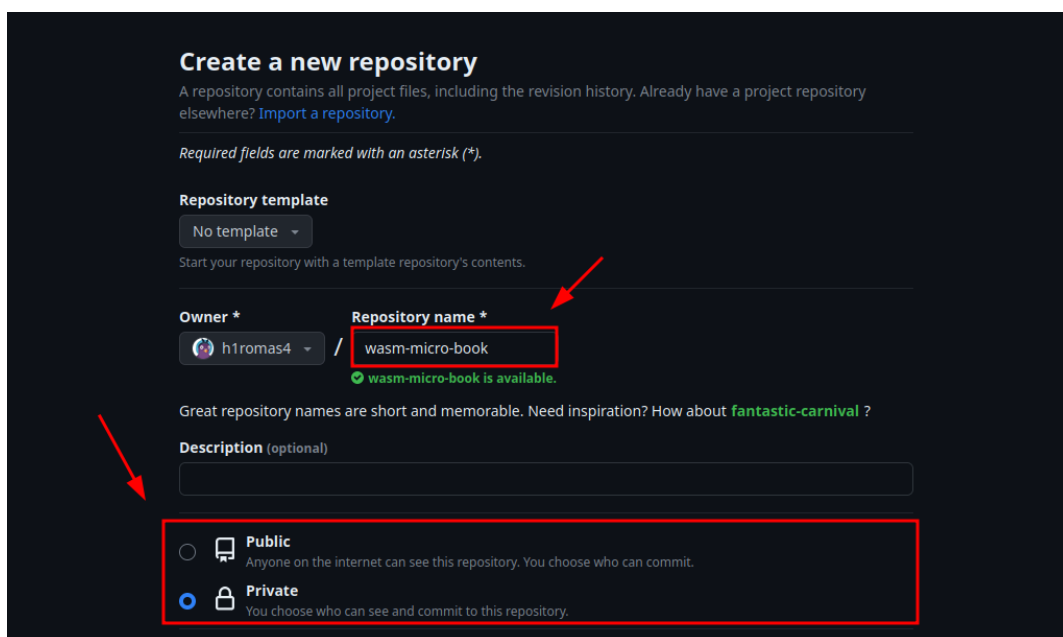


画面. GitHub の Repositories

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。

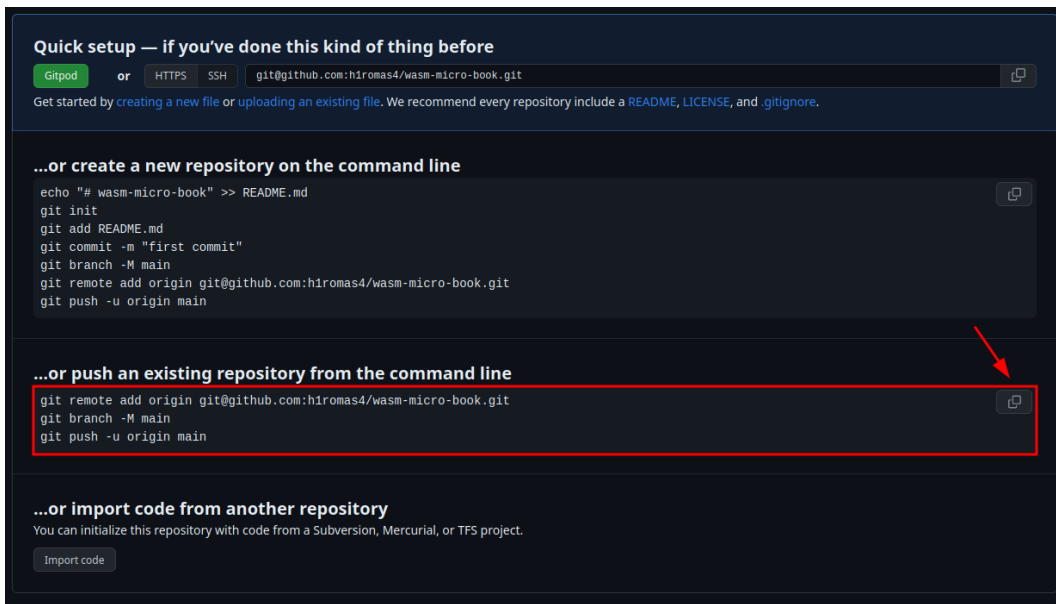


ここではリポジトリ名として `wasm-micro-book` を、公開範囲を執筆者のみが文書ファイルが閲覧・操作が可能となる **Private** を選択しました。なお、公開範囲に **Public** を指定した場合は、インターネット上の全てのユーザから文書ファイルの読み取りが可能になりますので注意します。



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ます。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。

手順. 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 ❶
git config --local user.email h1romas4@gmail.com ❷
git add .
git commit -m 'initial commit'
```

- ❶ 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。
- ❷ 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。

手順. 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 ❶  
git branch -M main  
git push -u origin main
```

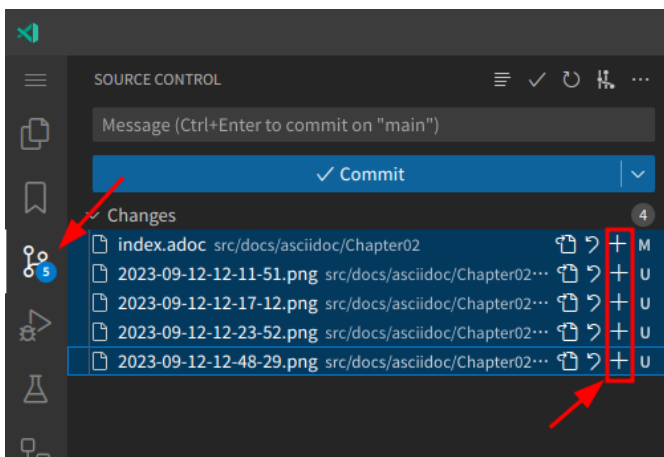
❶ h1romas4/wasm-micro-book.git 部分が前項で作成したリポジトリ名になります。

以上で GitHub リポジトリへの文書の配置が完了しました。ウェブブラウザで GitHub リポジトリにアクセスすると、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。

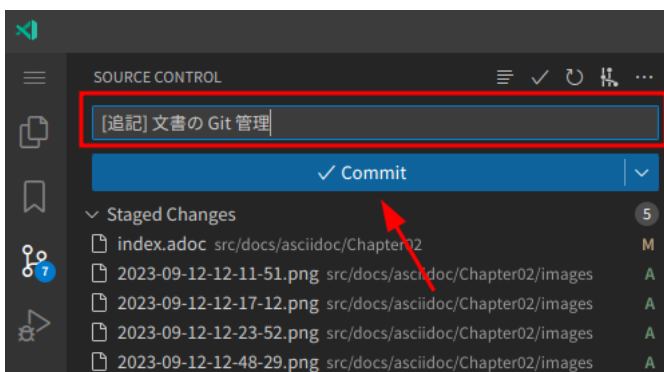
2.7.4. 修正したファイルのコミット・プッシュ

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

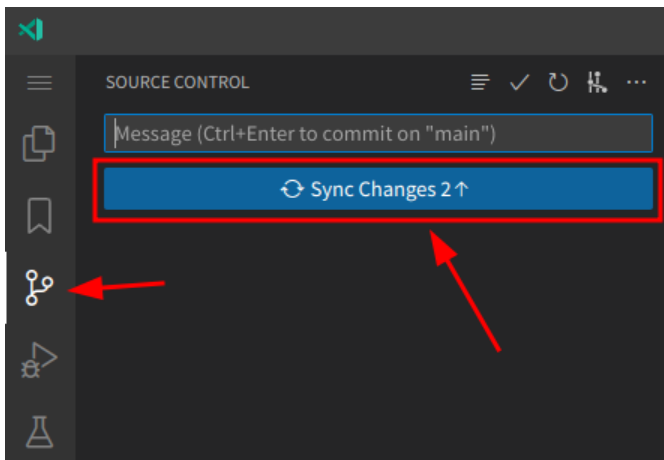


画面. VS Code から Git へのコミット

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。

手順. 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。

その環境で初めて執筆する場合は、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。

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

❶ `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。

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` 操作を行う。

手順. 統合ターミナルから `git pull` を実行

```
git pull
```

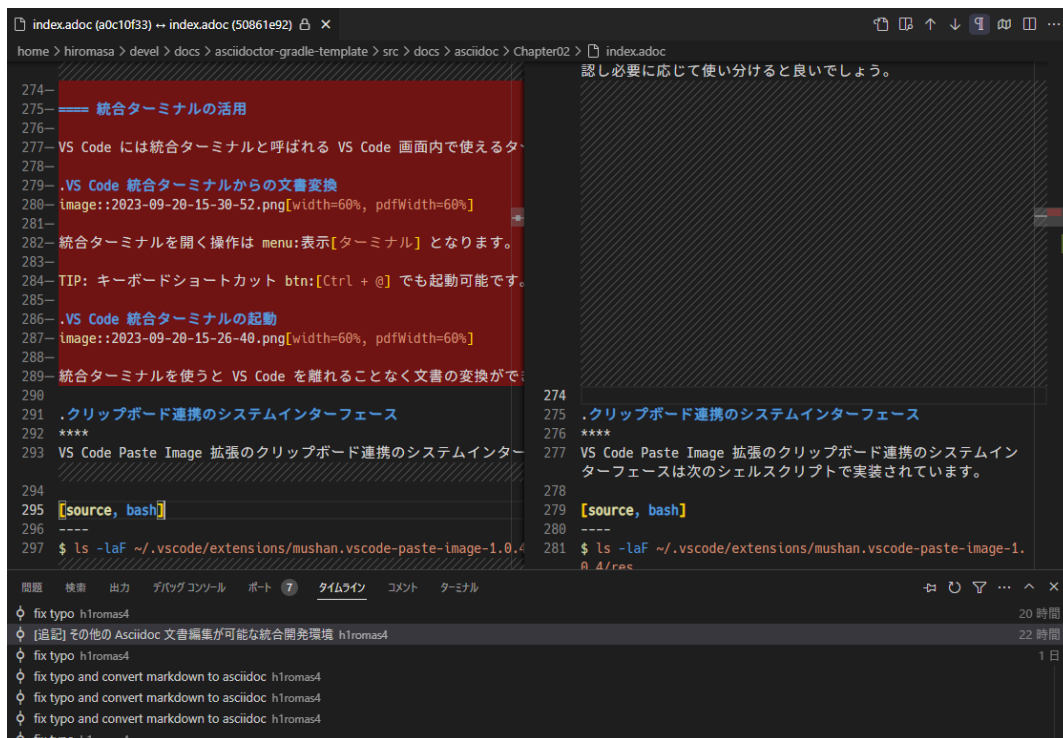


個人の執筆活動における Git 初歩としては「執筆開始前に必ず `git pull`」「執筆区切りで `commit` と `push`」と覚えることで十分機能するはずです。

2.7.6. Git による文書の差分比較

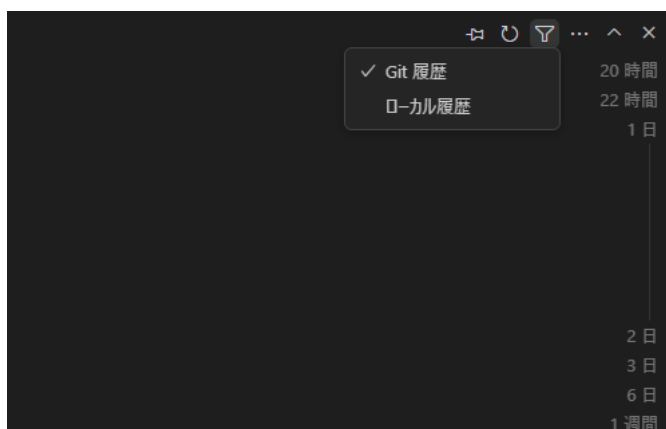
Git 管理された文書ファイルはコミットごとに履歴が取得され、各履歴ごとの差分比較が可能です。

VS Code では統合ターミナルと同列のパネルに配置されたタイムラインから履歴をダブルクリックすることで、Git コミットごとの修正前後の比較ができます。



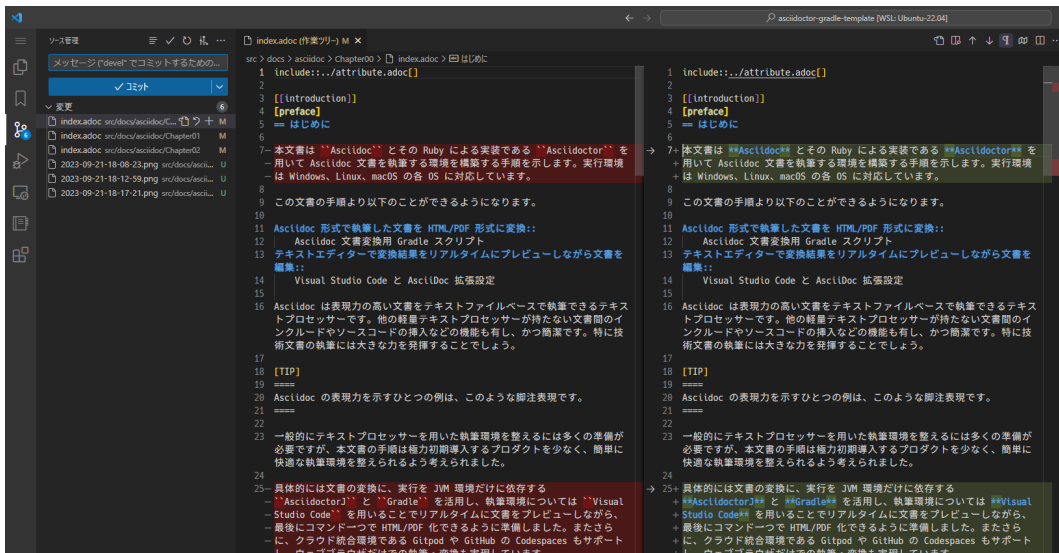
画面. VS Code タイムラインパネル

タイムラインには標準で Git の履歴とともに、ファイル上書き保存ごとの修正差分も表示されます。状況に応じてパネル右上のフィルター項目からそれぞれの表示可否を選択してください。



画面. VS Code タイムラインパネルのフィルター

また Git コミット時に使うソース管理パネルでも、ファイルをダブルクリックすることでコミット前後の差分が表示され修正内容を確認できます。



画面. VS Code ソース管理パネル



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が得られます。

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](https://www.gitpod.io/) や GitHub の [Codespaces](https://github.com/features/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 文書のビルドが可能です。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。

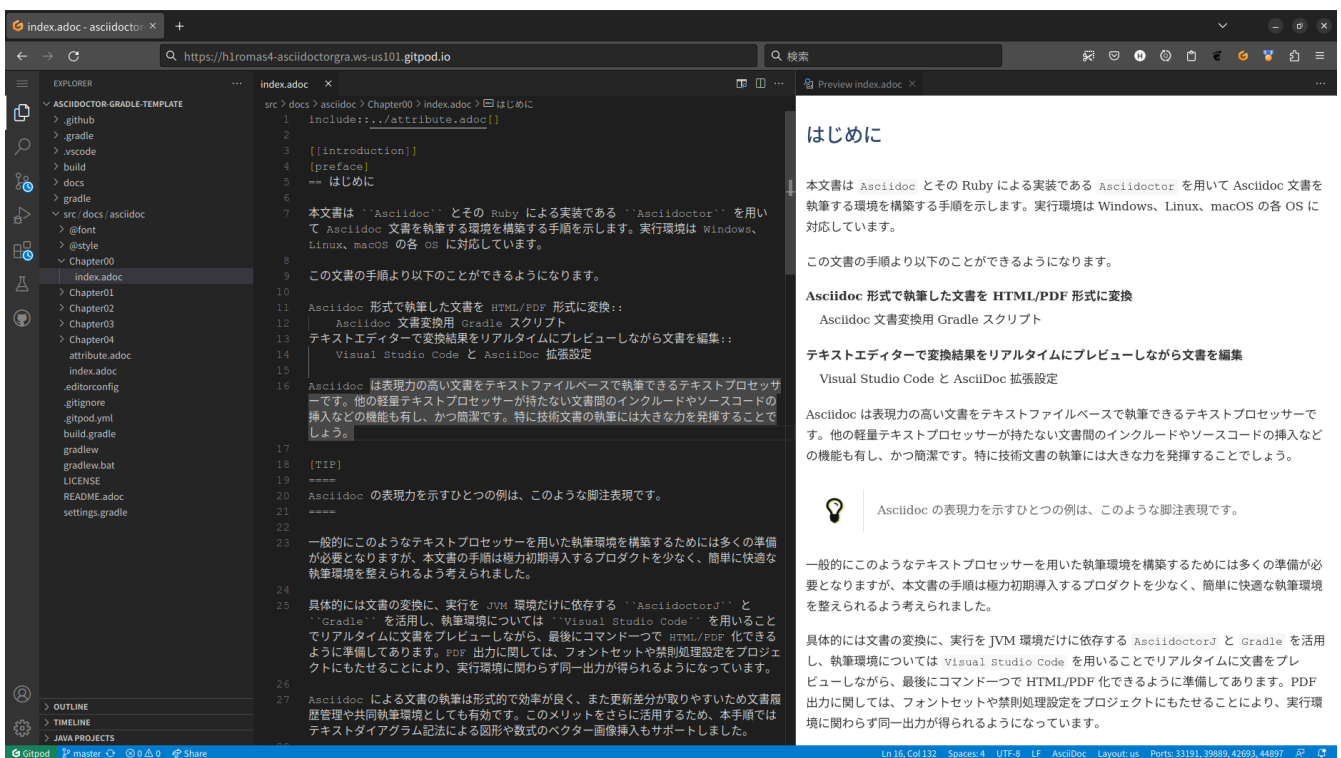
準備ができれば、次の要領で URL を作成しウェブブラウザでアクセスするとクラウド環境が起動します。

<https://gitpod.io/new/#https://github.com/h1romas4/asciidoctor-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/asciidoctor-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ないため Git への文書修正操作は試せません。



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文書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
#
```



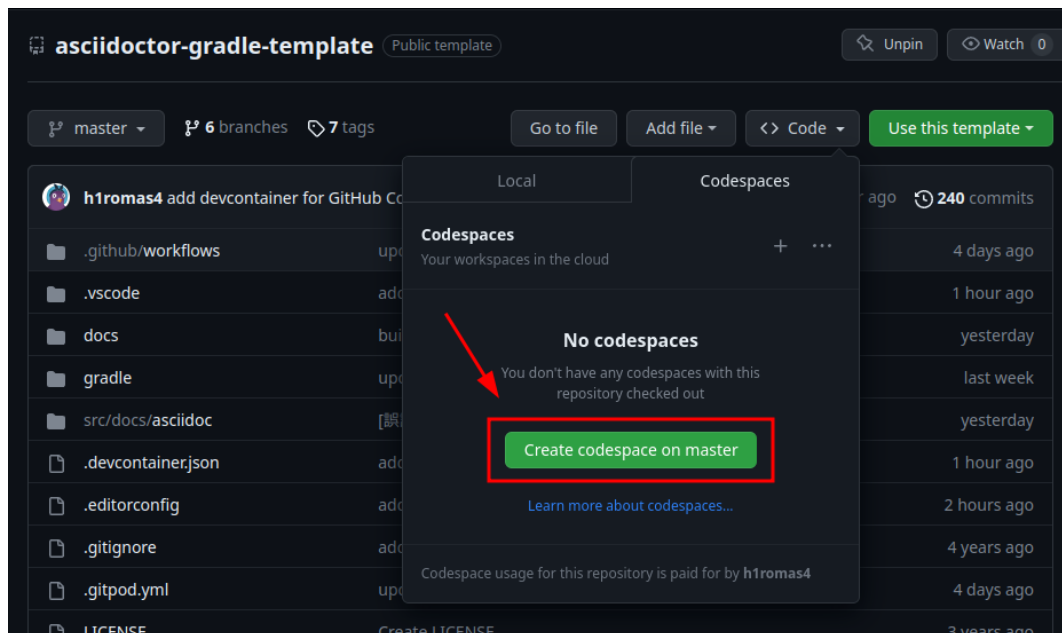
これらの Gitpod 向けの設定は `.gitpod.yml` ファイルで定義されています。

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。



<https://github.com/h1romas4/asciidoctor-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、文書の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-container.sh
```

文書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 ❶
$ fc-match Dialog
NotoSansCJK-Regular.ttc: "Noto Sans CJK JP" "Regular" ❷
$ ./gradlew docs ❸
# ..snip..
BUILD SUCCESSFUL in 3m 19s
```

- ❶ ja_JP.UTF-8 になっていること
- ❷ Noto Sans CJK JP になっていること
- ❸ 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、文書のビルド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統合ターミナルを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-container.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

クラウド環境上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にしてください。

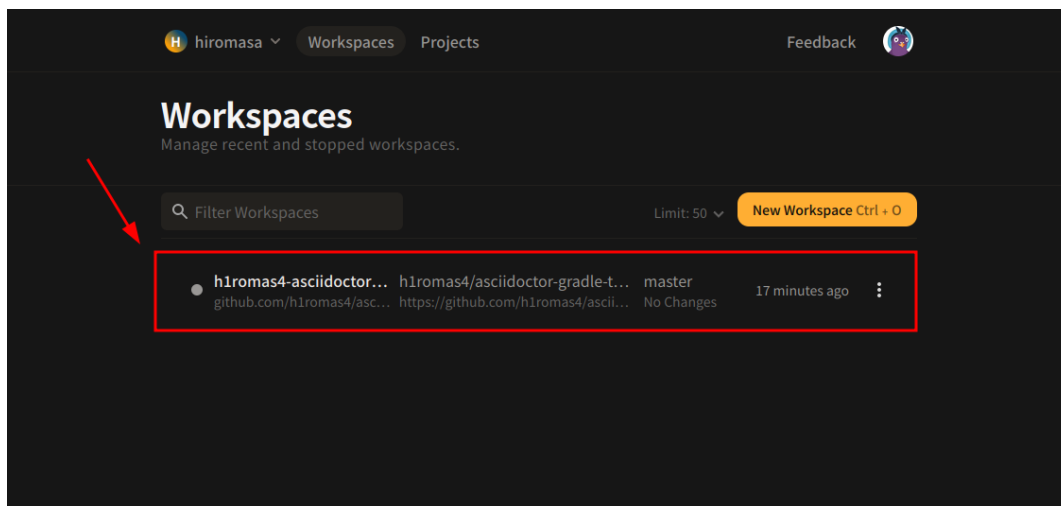


Git プッシュの操作を行わずにウェブブラウザを閉じてしまった場合はファイルが失われる可能性があります。十分注意してください。

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。

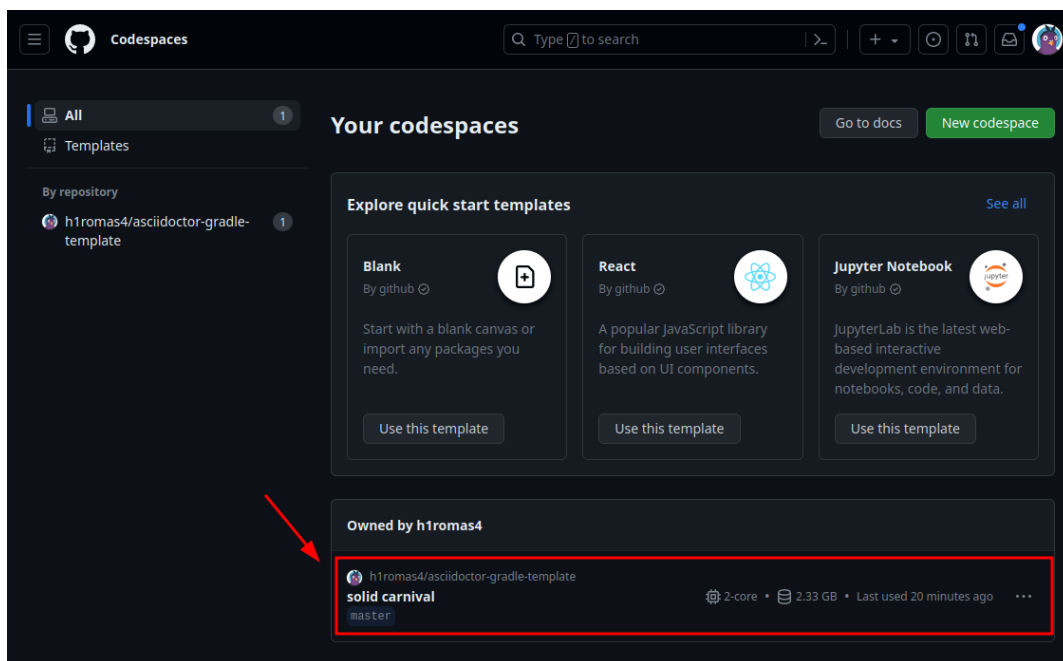
Gitpod の場合

<https://gitpod.io/workspaces>



Codespaces の場合

<https://github.com/codespaces>



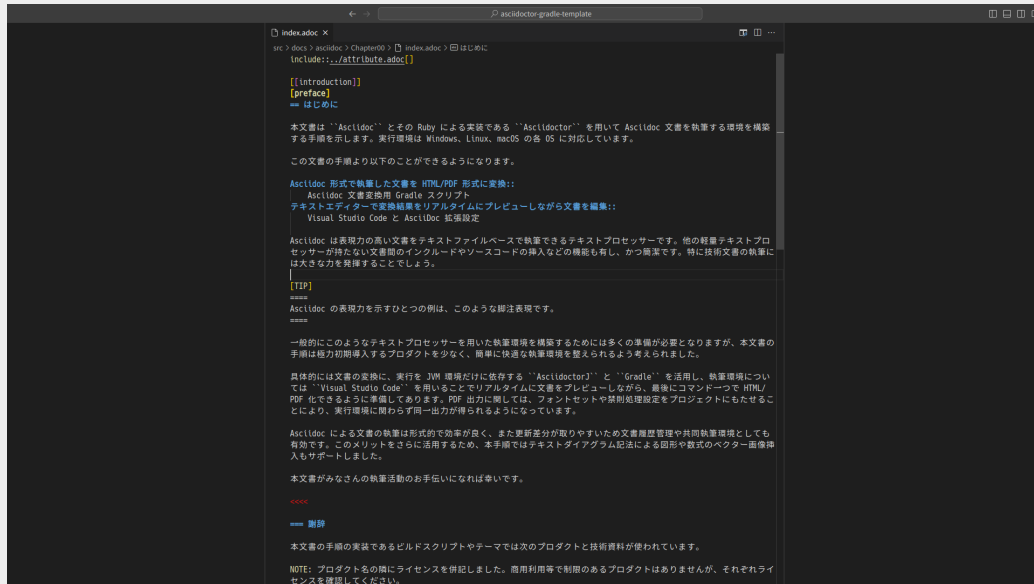
クラウド上で起動していた環境は最終利用時からある一定期間（数日）は残ります。この動作を利用して未コミット・プッシュのファイルを取り戻せます。

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ーメニューより、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、再度同じ操作もしくは **[Esc]** 2回押下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の良いツールとなることでしょう。

2.10. GitHub Actions によるビルド

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 サーバ上で文書のビルドテスト、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. 文書ファイルの変換テストと検査(gradle.yml)

git による main ブランチへの push 操作を契機に自動的に動作する文書変換テストです。macOS、Linux、Windows のそれぞれの環境で、Java 11/17/21 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 9 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、文書のビルド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 文書ファイルが生成されるか。
- 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$' ❶
    # test asciidoctor
    file docs/index.pdf | grep PDF
    file docs/index.html | grep HTML
```

- ❶ 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。

.github/workflows/gradle.yml

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

❶ PDF 文書のメタ情報を出力。

❷ pdftinfo コマンドで PDF 文書中に含まれる URL 文字列を抽出。



pdftinfo コマンド部分は Ubuntu ビルドのみ対応しています。また、メタ情報に関する可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。

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」を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](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 は次のようになっています。

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



2.10.2. 文書ファイルの自動リリース(release.yml)

git のバージョンタグを push する操作を契機として、GitHub のリリースページに GitHub Actions 上の環境で変換処理された HTML/PDF 文書をアーカイブファイルとして GitHub のダウンロードページにリリースします。

▼ Assets 8		
 asciidoctor-gradle-book-v4.0.0-by-macos-latest-java-11.tar.gz	4.39 MB	2 weeks ago
 asciidoctor-gradle-book-v4.0.0-by-macos-latest-java-17.tar.gz	4.39 MB	2 weeks ago
 asciidoctor-gradle-book-v4.0.0-by-ubuntu-22.04-java-11.tar.gz	4.37 MB	2 weeks ago
 asciidoctor-gradle-book-v4.0.0-by-ubuntu-22.04-java-17.tar.gz	4.37 MB	2 weeks ago
 asciidoctor-gradle-book-v4.0.0-by-windows-latest-java-11.tar.gz	4.38 MB	2 weeks ago
 asciidoctor-gradle-book-v4.0.0-by-windows-latest-java-17.tar.gz	4.38 MB	2 weeks ago
 Source code (zip)		2 weeks ago
 Source code (tar.gz)		2 weeks ago

画面. GitHub のリリースページに掲載された変換後の文書

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。

.github/workflows/release.yml

```
jobs:
  build:
    strategy:
      matrix:
        os: [ ubuntu-24.04, windows-latest, macos-latest ] ❶
        java: [ '11', '17', '21' ] ❷
```

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

❷ 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 11/17/21 の計 9 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付与されるバージョン番号については、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。

.github/workflows/release.yml

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

- ❶ asciidoctor-gradle-book 部分が出力されるファイル名に前置されます。
- ❷ Git のタグ名がそのままバージョン名となります。

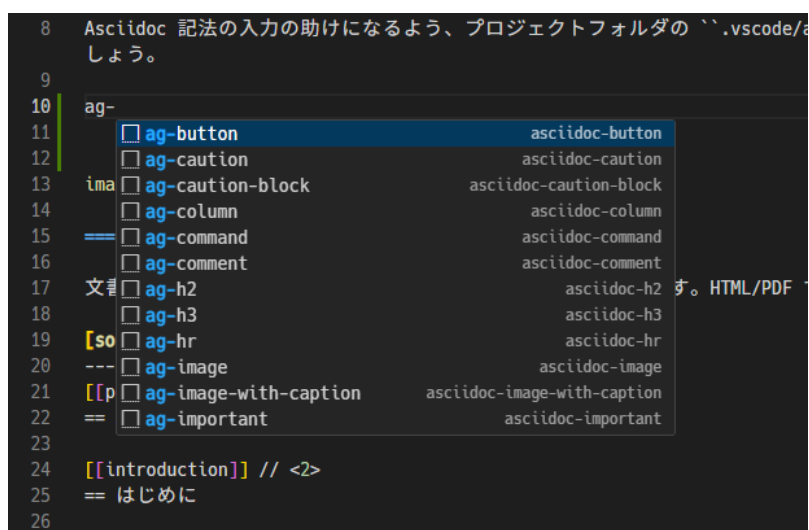
本リリースの機能を使い頒布や出版時にタグを打つことで、その時点のファイル断面がダウンロード可能な形で永続化できます。文書の版管理に活用してください。

3. AsciiDoc 記法

AsciiDoc ではテキスト中に特定の記法を挿入することで文書表現や付随情報を追加しながら執筆を進めます。

本項では AsciiDoc 文書で利用可能な記法の中で、この文書で使われている表現を取り上げて解説します。初めは本文書を手元に置いて、使いたい部分からコピーアンドペーストするのも良い方法です。

また、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。



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. ドキュメントヘッダーとセクションタイトル

文書名を表すドキュメントヘッダーや、章やチャプターを表すセクションタイトルは次のように記述します。

= AsciiDoctorとGradleでつくる文書執筆環境 ❶

== はじめに ❷

❸

本文書は AsciiDoc とその Ruby による実装である... ❹

- ❶ ドキュメントヘッダー
- ❷ セクションタイトル
- ❸ 本文との間には空行を入れる。
- ❹ ここから本文。



本文書で提供される `.adoc` はドキュメントタイプが `book` に指定され、`=`（レベル 0）がドキュメントヘッダー（文書名）となり、それ以降がセクションタイトルとなります。
 なお、ドキュメントタイプは `attribute.adoc` の `:doctype: book` で指定されています。

3.2. 段落

文章の段落は改行でコントロールします。新しい段落にする場合は改行のみの空行を挿入します。

こんにちは。

①

吾輩は段落である。（ここには空行はない） ②
 名前はまだない。

① 空行により「こんにちは。」が段落となる。

② 空行がない行は同一段落として扱われる。

こんにちは。

吾輩は段落である。（ここには空行はない） 名前はまだない。

3.3. 文章中のテキストフォーマット

3.3.1. コマンドやファイル名などの等角フォント

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``
```

3.3.2. 強調

文中の**強調**は `**` で括ることで記述します。

文中の****強調****は ```**``` で括ることで記述します。

3.3.3. エスケープ

文章中に AsciiDoc 記法で使われる文字列と同一の文字が存在し、意図せず AsciiDoc として解釈されてしまう場合は、`\` でエスケープすることができます。

…参照リンクは参照先に ``[[ラベル名]]`` を付与して…

…参照リンクは参照先に [[ラベル名]] を付与して…

3.4. 文書中の相互参照

.adoc 文書内の参照リンクは参照先に [[ラベル名]] を付与して設定します。HTML/PDF 上はクリックリンクが生成されます。

```
[[project-structure]] ❶  
== AsciiDoc 記法  
  
[[introduction]] ❷  
== はじめに  
  
<<project-structure,章の冒頭>> ❶  
<<Chapter01/index.adoc#introduction,はじめに>> ❷
```

❶ 同一 .adoc 内での内部参照用のラベル定義と参照

❷ ファイルを跨いだ場合のラベル参照

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…

3.5. 外部リンク

インターネット上の URL 文字列は自動的にリンクに変換されます。

<https://h1romas4.github.io/asciidoctor-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/asciidoctor-gradle-template/index.html[AsciidoctorとGradleでつくる文書執筆環境]
```

[Asciidoctor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。

3.6. 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, AsciiDoctor Documentation Site]
```

```
----
```

```
https://docs.asciidoctor.org/
```

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

```
----
```

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

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

— AsciiDoctor Documentation Site

3.7. 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 文書中で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。



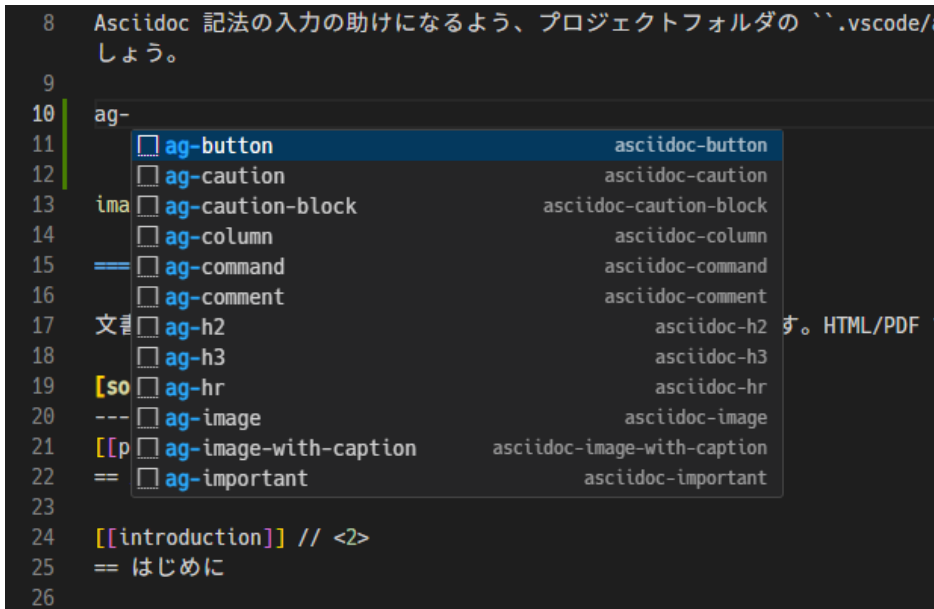
これは `src/docs/asciidoc/attribute.adoc` で `:imagesdir: ./images` が設定済みであるための動作です。

また、ブロックタイトルと画像を組み合わせることで画像キャプションを追加できます。

.VS Code の IntelliSense ❶
 [caption="画像. "] ❷
 image::2023-09-14-18-52-05.png[pdfwidth=70%, width=70%]

❶ ブロックタイトルを指定

❷ キャプションを指定



画像. VS Code の IntelliSense

3.8. 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() { ❶
    console.log("Hello, World!"); ❷
}
```

- ❶ 関数定義
- ❷ コンソールに Hello, World を出力

3.9. 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
```



この表現は AsciiDoc のマクロで実装されており、`attribute.adoc` の `:experimental:` (実験的) 指定で有効にされています。

3.10. 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` と `clean` タスクを起動し `docs` フォルダをクリーニングしてから再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!");
}
```

3.11. 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、罫線表現に関していくつかの属性が設定できます。

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



cols="1,2" 指定することでカラムの比率を 1:2 に設定しています。

表形式の例2 (ヘッダーあり、グリッドあり、ストライプなし)

```
[cols="1,2", options="header", stripes="none"]
[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

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	1. リスト1 2. リスト2

3.12. リスト

箇条書き表現は次のように記述します。



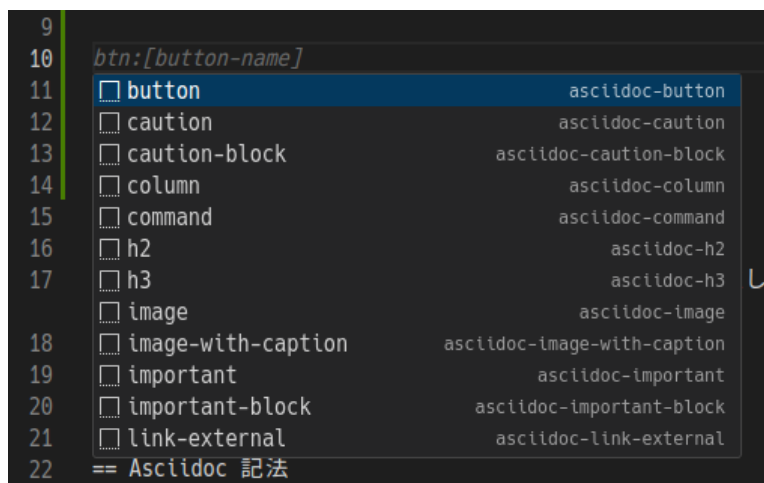
AsciiDoc はリスト中にリスト継続マーカ (+) を使うことでブロックのネストが可能です。手順を順序ありリストで表現し、順序やインデントを維持しながら画像や表形式などの挿入をします。

```
. 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



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

3.13. 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。

3.14. 改ページ・ブレイク表現

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

```
<<<
```

ブレイク表現（水平線）は HTML/PDF 文書のどちらでも有効です。

```
'''
```

3.15. ラベル文言

特定のラベル文言に対して定義を明記する表現は次のように記述します。

```
対応 OS::
    macOS、Linux、Windows 対応
対応 JDK::
    11、17、21 の各 LTS バージョン
```

対応 OS

macOS、Linux、Windows 対応

対応 JDK

11、17、21 の各 LTS バージョン

3.16. コラム表現

AsciiDoc のサイドバーとブロックタイトル記法を組み合わせ、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。

3.17. 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

ダイアグラム記法は、図表をソースコードから出力できる特別なマークアップです。AsciiDoc と同様に文書ファイル中のテキスト形式で "描ける" ことから、修正や差分管理がしやすいという利点があります。

本項では **PlantUML** と **dittaa** 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例を示します。

文書のビルドではダイアログ記法をサポートする **asciidoctorj-diagram** が有効になっており、PlantUML と ditaa ダイアグラム記法を追加の操作無しで使えます。また、PDF 出力時に日本語が化けないように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。

AsciiDoc 文書内でダイアグラムを SVG ベクター画像で出力する指定は次のようになります。

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

```
----
```

```
// ダイアグラム記法のソースコード
```

```
----
```

- ❶ ダイアグラム種類には `plantuml` か `dittaa` を指定する。
- ❷ `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) で指定する。



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アンド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。

次項から PlantUML や ditaa 記法を使って描いた実際のダイアグラム図表とソースコードを紹介します。コピーアンドペースト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得られます。

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

PlantUML in a nutshell

— PlantUML

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

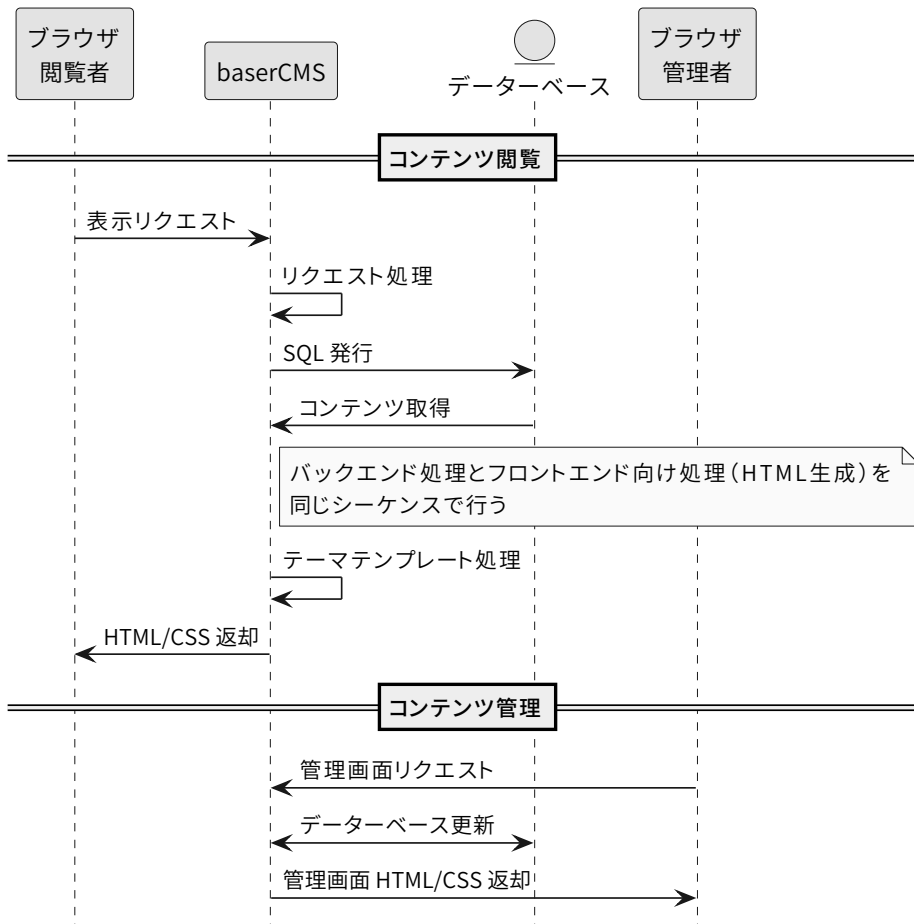
Diagrams Through Ascii Art by Stathis Sideris

— 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 による文書のビルドでは内蔵の PlantUML ライブラ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 ❶
}
</style>
{
{T
+ <&folder> wp-admin
}
}
@endsalt
```

❶ 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。

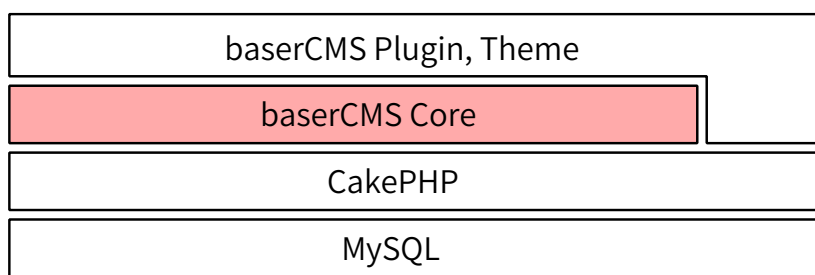


ditaa 内で日本語を使う場合は、表示ずれを防ぐため半角スペースなどで文字数を調整する必要があります。また、ドロップシャドウ shadows 設定は HTML 版に対してのみ有効で、現在のところ PDF 版に対しては効果がないようです。

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

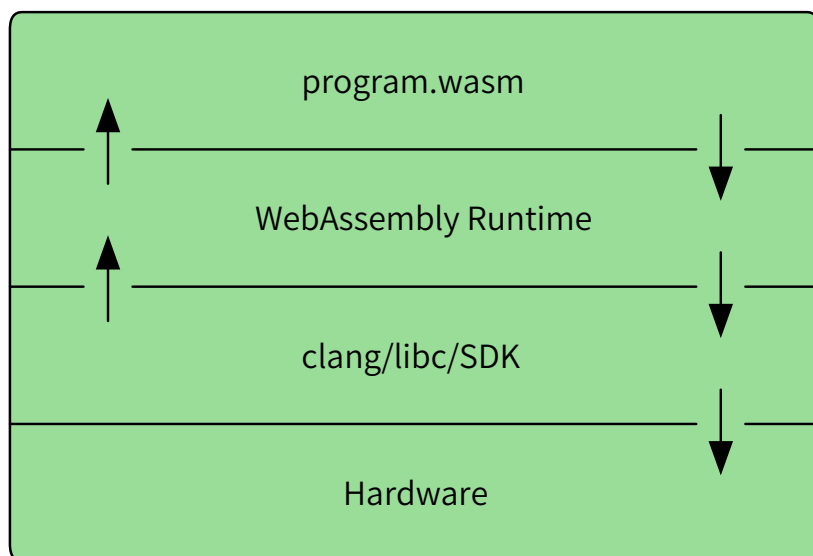
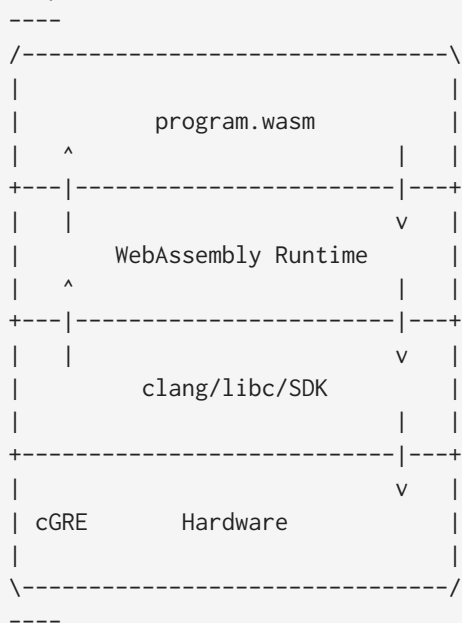
```
----
+-----+
|   baserCMS Plugin, Theme   |
+-----+ +
| cPNK   baserCMS Core      | |
+-----+ +-----+
|               CakePHP     | |
+-----+ +-----+
|               MySQL       | |
+-----+ +-----+

/-----+-----\
|cRED RED   |cBLU BLU   |
+-----+ +-----+
|cGRE GRE   |cPNK PNK   |
+-----+ +-----+
|cBLK BLK   |cYEL YEL   |
\-----+-----/
----
```



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

```
[caption=""]
```

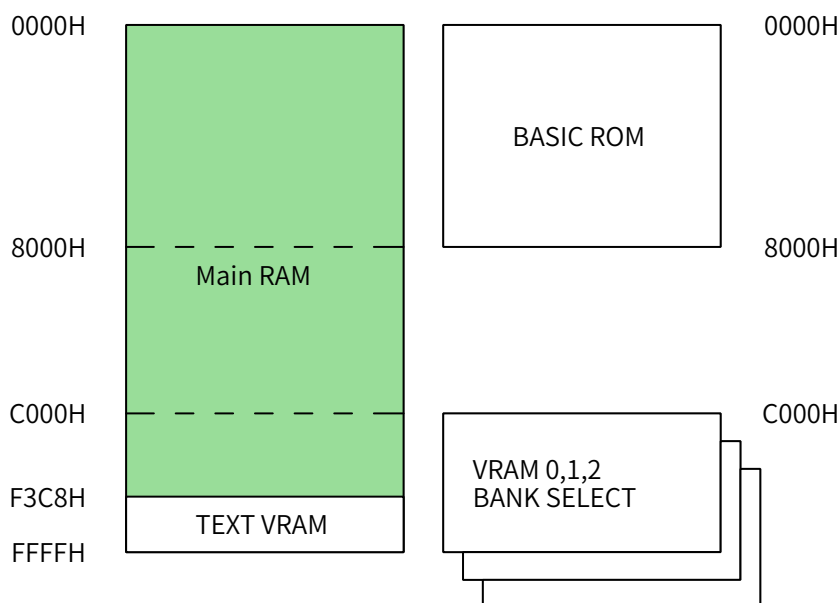


4.5. メモリーマップ

メモリーマップを ditaa 記法で記述します。

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

```
-----  
0000H +-----+ +-----+ 0000H  
      | cGRE      | |      |  
      |          | |      |  
      |          | |      |  
      |          | |      |  
      |          | |      |  
      |          | |      |  
8000H +-----+ +-----+ 8000H  
      | Main RAM  | |      |  
      |          | |      |  
      |          | |      |  
      |          | |      |  
C000H +-----+ +-----+ C000H  
      |          | |      | ++  
      |          | | VRAM 0,1,2 | ++  
F3C8H +-----+ | BANK SELECT | |  
      | TEXT VRAM | |          | |  
FFFFH +-----+ ++-----+ |  
                      ++-----+ |  
                      +-----+  
-----
```



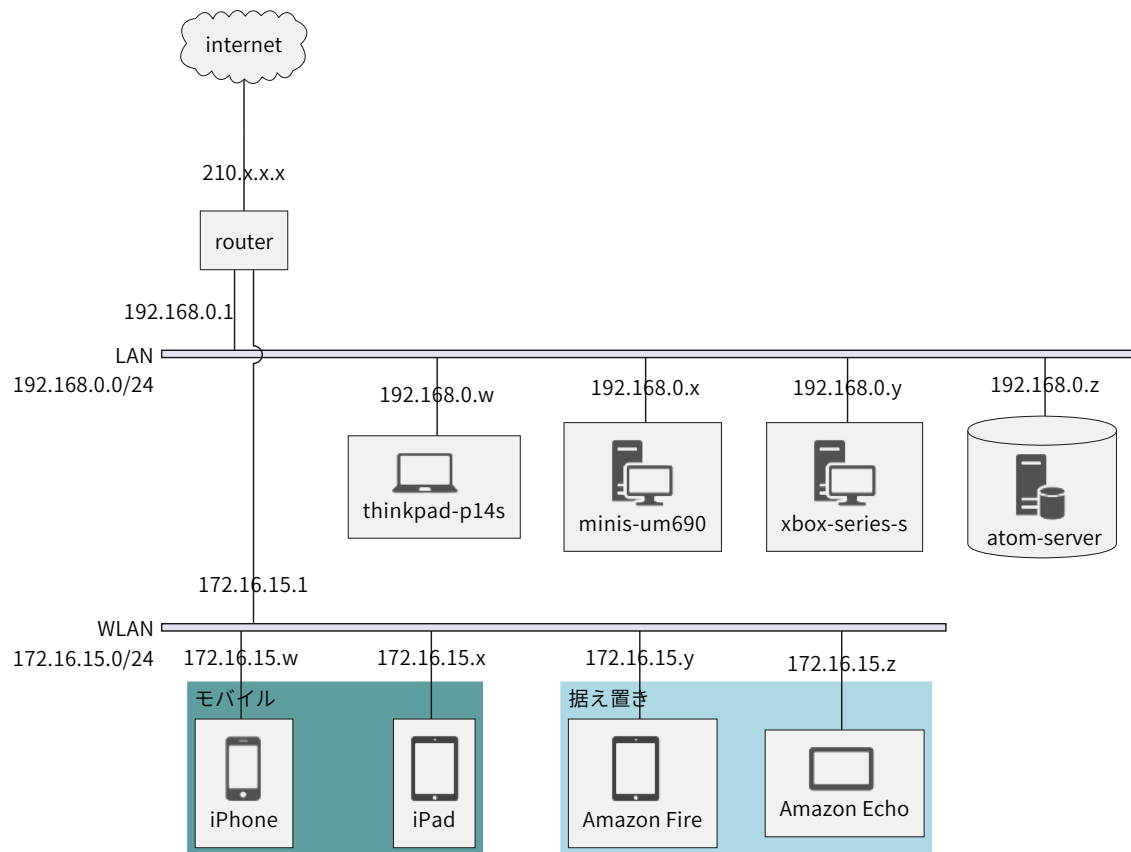
4.6. ネットワーク構成図

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_STEREOYPE()

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, "Bacth 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_admnin, "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_admnin, 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())
```



```

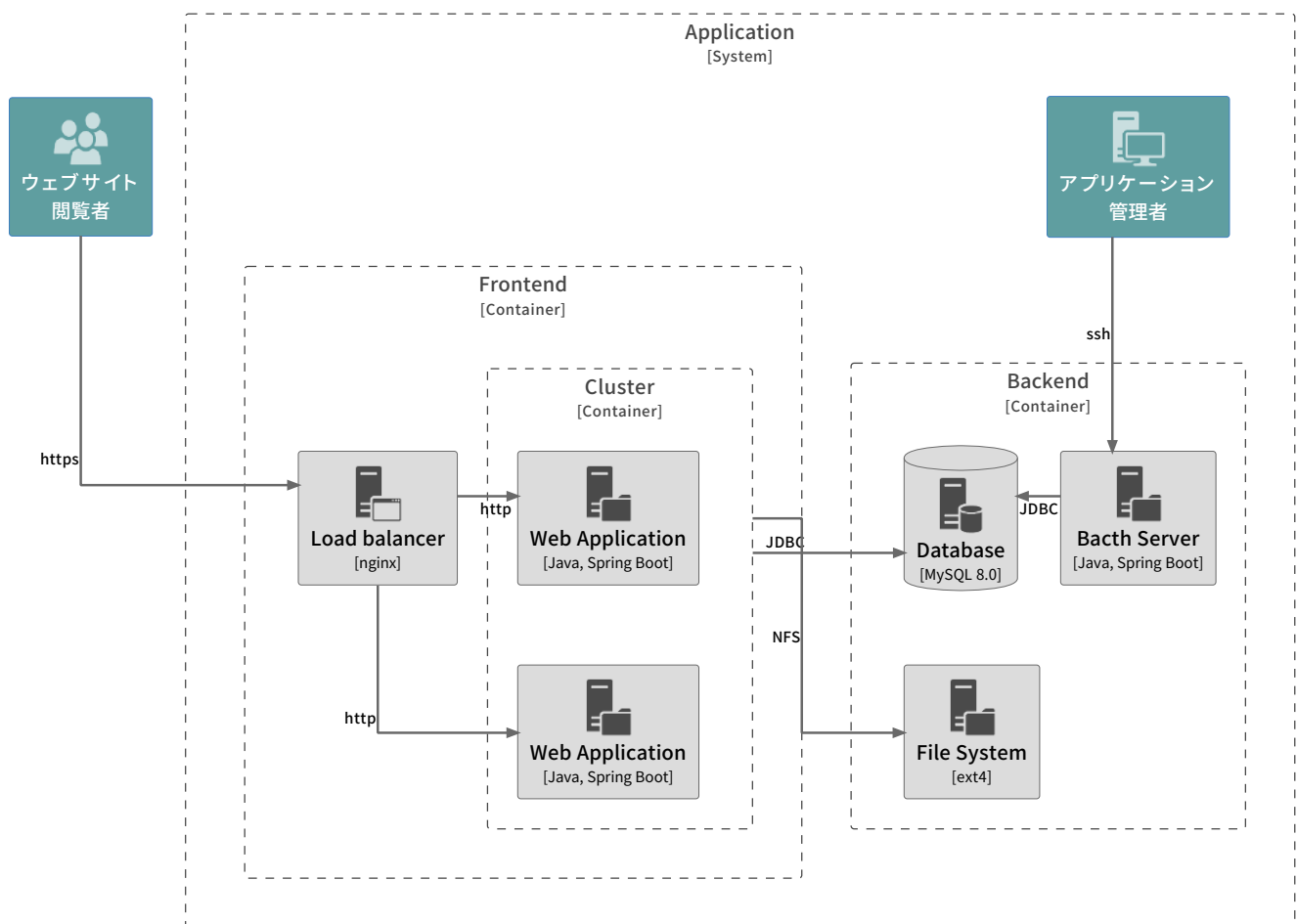
Rel_R(user_person, load_balancer, "https", $tags="rel_line")
Rel_R(load_balancer, web_app_1, "http", $tags="rel_line")
Rel_R(load_balancer, web_app_2, "http", $tags="rel_line")

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_adminin, batch, "ssh", $tags="rel_line")

@enduml
-----

```



現在のところ \$shadowing="true" 指定のドロップシャドウは 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

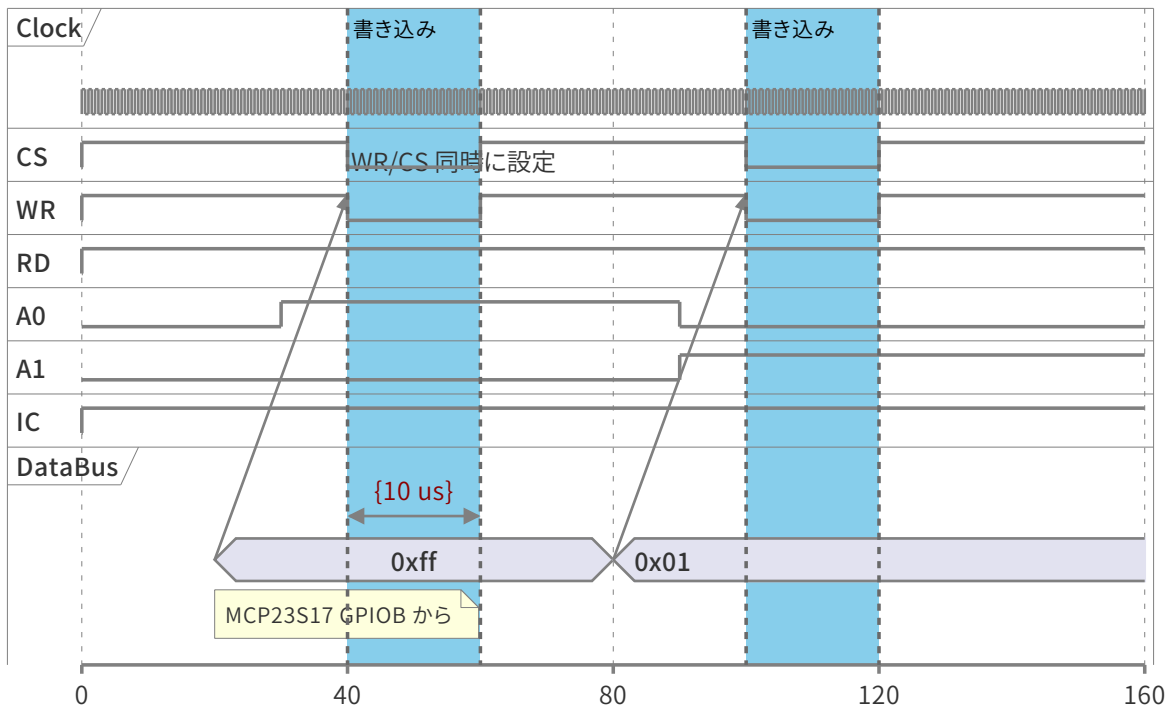
@:set_data_bus_1
DB is "0xff"
DB -> WR@+20
note bottom of DB : MCP23S17 GPIOB から

@: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

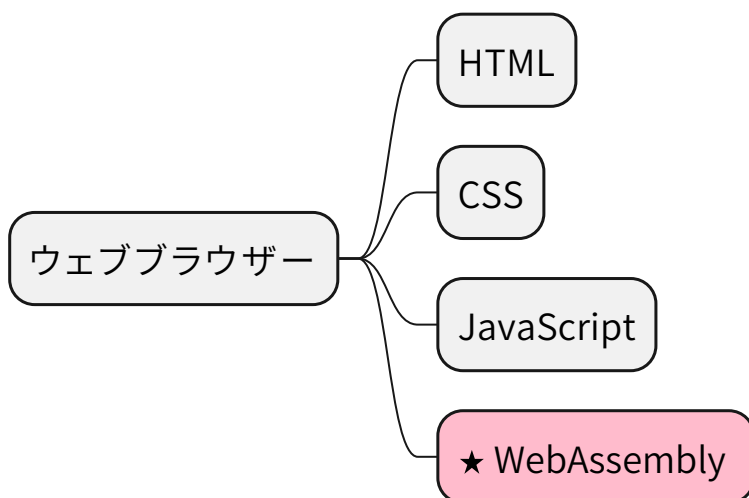
PlantUML 内で `<アイコン名>` 形式で使える Open Iconic アイコンは次の通りです。なお、アイコン名はこの文書が HTML 版の場合 [ウェブブラウザ](#) > [右クリック](#) > [画像を開く](#) するとコピーアンドペーストで取得できます。

List Open Iconic	▲ bell	☁ cloud	≡ excerpt	≡ justify-right	🎵 musical-note	★ star
Credit to	📶 bluetooth	☁ cloudy	⌵ expand-down	🔑 key	📎 paperclip	☀ sun
https://useiconic.com/open	B bold	🔗 code	⌵ expand-left	💻 laptop	🖋 pencil	📱 tablet
↶ account-login	⚙ bolt	🧠 cog	⌵ expand-right	📶 layers	👤 people	🏷 tag
↷ account-logout	📅 book	⌵ collapse-down	⌵ expand-up	💡 lightbulb	👤 person	🏷 tags
↶ action-redo	🔖 bookmark	⌵ collapse-left	🔗 external-link	🔗 link-broken	📱 phone	🎯 target
↶ action-undo	📦 box	⌵ collapse-right	👁 eye	🔗 link-intact	📊 pie-chart	📋 task
≡ align-center	👜 briefcase	⌵ collapse-up	🧴 eyedropper	≡ list-rich	📌 pin	🖨 terminal
≡ align-left	🌍 british-pound	✎ command	📁 file	≡ list	🎮 play-circle	⌨ text
≡ align-right	🔍 browser	📄 comment-square	🔥 fire	📍 location	⊕ plus	👇 thumb-down
🔍 aperture	🖌 brush	🧭 compass	🚩 flag	🔒 lock-locked	🔌 power-standby	👍 thumb-up
↓ arrow-bottom	🐛 bug	🔍 contrast	⚡ flash	🔓 lock-unlocked	🖨 print	🕒 timer
🕒 arrow-circle-bottom	📣 bullhorn	≡ copywriting	📁 folder	🔄 loop-circular	📁 project	⇄ transfer
🕒 arrow-circle-left	📊 calculator	📇 credit-card	🍴 fork	🔄 loop-square	✂ pulse	🗑 trash
🕒 arrow-circle-right	📅 calendar	📊 dashboard	🖥 fullscreen-enter	🔄 loop	🧩 puzzle-piece	📏 underline
🕒 arrow-circle-top	📷 camera-slr	⬇ data-transfer-download	🖥 fullscreen-exit	🔍 magnifying-glass	❓ question-mark	⬇ vertical-align-bottom
↶ arrow-left	📶 caret-bottom	⬆ data-transfer-upload	🌐 globe	📍 map-marker	☔ rain	⬆ vertical-align-center
→ arrow-right	⏪ caret-left	🗑 delete	📊 graph	🗺 map	🎲 random	⬆ vertical-align-top
↓ arrow-thick-bottom	⏩ caret-right	🗑 delete	📊 grid-four-up	📺 media-pause	🔄 reload	📺 video
↶ arrow-thick-left	⏪ caret-top	📄 dial	📊 grid-three-up	▶ media-play	↕ resize-both	🔊 volume-high
→ arrow-thick-right	🗨 chat	📄 document	📊 grid-two-up	📄 media-record	⬆ resize-height	🔊 volume-low
↑ arrow-thick-top	✓ check	💵 dollar	📊 hard-drive	⏮ media-skip-backward	↔ resize-width	🔊 volume-off
↶ arrow-top	✓ chevron-bottom	” double-quote-sans-left	📊 header	▶ media-skip-forward	📡 rss-alt	⚠ warning
📊 audio-spectrum	✓ chevron-left	” double-quote-sans-right	📊 headphones	⏮ media-step-backward	📡 rss	📶 wifi
🔊 audio	⏪ chevron-right	” double-quote-serif-left	♥ heart	▶ media-step-forward	📄 script	🔧 wrench
🏷 badge	➡ chevron-top	” double-quote-serif-right	🏠 home	■ media-stop	📦 share-boxed	✕ x
🚫 ban	🔍 cloud-download	💧 droplet	🖼 image	🔍 medical-cross	📦 share	¥ yen
📊 bar-chart	🕒 circle-check	🚀 eject	📧 inbox	≡ menu	🛡 shield	🔍 zoom-in
📺 basket	🕒 circle-x	🛡 elevator	∞ infinity	🔊 microphone	📶 signal	🔍 zoom-out
🔋 battery-empty	📄 clipboard	📄 ellipses	📶 info	⏮ minus	📶 signpost	
🔋 battery-full	🕒 clock	📄 envelope-closed	📶 italic	📺 monitor	📶 sort-ascending	
🔋 beaker	☁ cloud-upload	📄 envelope-open	≡ justify-center	🌙 moon	📶 sort-descending	
		€ euro	≡ justify-left	➡ move	📊 spreadsheet	

4.9. マインドマップ

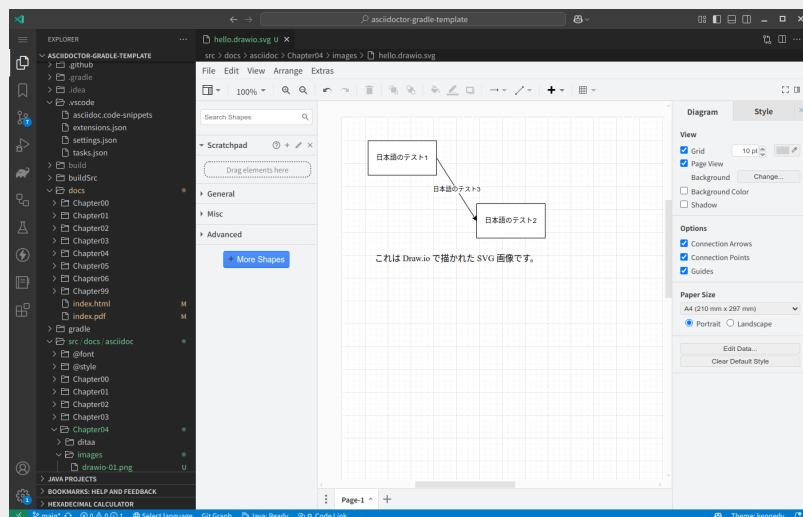
PlantUML のマインドマップ記法を使い木構造を表現します。

```
[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
----
```

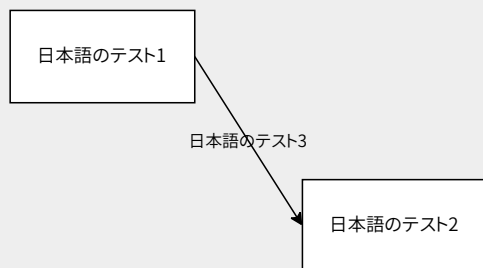


Draw.io でのダイアグラム作成

Draw.io はオープンソースで提供されるウェブブラウザ上で任意のベクター画像を描画できるツールです。VS Code では `hediet.vscode-drawio` 拡張を導入することで、次のように VS Code 上でオフライン Draw.io エディタが使用可能です。

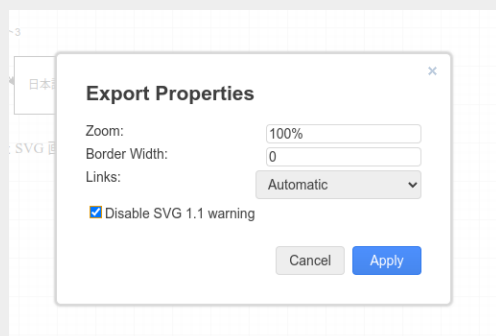


本手順では Draw.io で作成した SVG ファイルを PDF 文書に埋め込んだ際に日本語が文字化けしないように事前設定がされています。Draw.io で図形編集する場合はテキストフォントを標準から変更せずに Helvetica を維持してください。



これは Draw.io で描かれた SVG 画像です。

また、もし PDF 文書に挿入した SVG 画像に `Text is not SVG - cannot display` という文字がオーバーレイされてしまう場合は、**File > Properties** から **[Disable SVG 1.1 warning]** にチェックをつけファイルを保存し直してください。



5. Gradle ビルドの詳細

本項では、これまで使ってきた docs や clean のような Gradle タスクの詳細を解説します。この内容は執筆に直接影響しませんが、予備知識として知ることによってプロジェクトフォルダ構成変更などビルドのカスタマイズが行えます。

また合わせて、ここまで登場しなかった執筆中に有用と思われるユーティリティタスクについても示します。

本項で解説する Gradle タスクは次のとおりです。

Gradle タスク一覧

タスク名	役割
tasks	build.gradle で定義された全てのタスク名と説明を出力します。
docs	src/docs/asciidoc/index.adoc の Asciidoc 文書を HTML/PDF 文書に変換し docs フォルダに格納します。
clean	build フォルダ配下のファイルを削除しビルドを初期化します。 include 先ファイルの更新や PDF テーマの修正など、.adoc 自体に変更がない場合に本タスクを実行後、再ビルドすると結果が反映されます。
cleanDocs	docs フォルダ配下のファイルを全て削除します。 docs タスクの前提依存となっているため、このタスクを直接実行する機会はほとんどありませんが、ビルドスクリプトのカスタマイズを行う場合に使えるかもしれません。
checkPdfInfo	追加のユーティリティタスクです。 docs/index.pdf 文書から PDF ファイルに設定されたメタ情報を出力します。このタスクは PDF 文書に誤った情報が混入していないかの確認に使えます。
checkSyncImage	追加のユーティリティタスクです。 src/docs/asciidoc 内にある画像ファイルと Asciidoc 文書からリンクされている画像リソースを比較し、リンクされていない画像やリンク切れファイルを出力します。このタスクは過去に使われ必要なくなった画像ファイルを検出するのに便利です。

5.1. Gradle タスクの実行方法

各々のタスクは build.gradle ファイルで定義され、gradlew シェルスクリプト（Windows では gradlew.bat）からタスク名を指定することで処理が実行されます。

手順. Gradle タスクの実行(macOS / Linux)

```
./gradlew docs ❶
```

- ❶ docs 部分がタスク名。

手順. Gradle タスクの実行(Windows)

```
.\gradlew docs ❶
```

- ❶ docs 部分がタスク名。gradlew 部分は拡張子省略形で、実際に動作しているのは gradlew.bat ファイルとなる。



Windows PowerShell 環境では、パスの区切り文字を \ ではなく、macOS や Linux と同様に / としても正しく動作します。そこで、ここからの解説では環境による操作の差異を省くため、パス区切りを / に表記として示します。また、各ユーティリティタスクが出力するパスも / 形式に統一されています。

5.2. tasks タスク

Gradle が持つ全てのタスクは tasks タスクにより一覧表示できます。

文書のビルドで使われるタスクは Documentation tasks グループに纏まって出力されます。本タスクは詳細なタスク名を忘れた場合に使うと便利です。

手順. gradlew tasks の実行

```
$ ./gradlew tasks
..snip..
Documentation tasks
-----
asciidoctor - Generic task to convert AsciiDoc files and copy related resources
asciidoctorPdf - Convert AsciiDoc files to PDF format
checkPdfInfo - Utility task to check PDF document metadata
checkSyncImage - Utility to check the consistency of AsciiDoc text and file system images
cleanDocs - Task to delete all files and directories under docs/
docs - Generate HTML/PDF documents from AsciiDoc documents and output them under docs/
..snip..
```

5.3. docs タスク

src/docs/asciidoc 配下の AsciiDoc 文書を HTML/PDF 文書に変換し docs フォルダに配置するタスクです。

手順. docs タスク

```
./gradlew docs
```

実際の動作としては Asciidoctor Gradle Plugin の `asciidoctor` 及び `asciidoctorPdf` タスクが文書変換処理を担います。

`asciidoctor` 及び `asciidoctorPdf` タスクの定義は次のとおりです。

`build.gradle` の `asciidoctor` タスク定義

```
asciidoctor {
    dependsOn asciidoctorGemsPrepare ❶
    baseDir file('src/docs/asciidoc') ❷
    sources {
        include 'index.adoc' ❸
    }
    resources {
        from('src/docs/asciidoc') {
            include 'Chapter*/images/*' ❹
        }
    }
    asciidoctorj {
        attributes 'stylesdir': '@style',
                  'stylesheet': 'asciidoctor.css' ❺
    }
}
```

- ❶ 追加の Ruby スクリプトを最初にロードします。
- ❷ Asciidoc 文書の起点フォルダです。
- ❸ Asciidoc 文書の起点ファイルです。
- ❹ 画像リソース定義です。
- ❺ HTML 文書用のスタイル CSS 定義です。

`build.gradle` の `asciidoctorPdf` タスク定義

```
asciidoctorPdf {
    dependsOn asciidoctorGemsPrepare ❶
    baseDir file('src/docs/asciidoc') ❷
    fontsDir file('src/docs/asciidoc/@font') ❸
    sources {
        include 'index.adoc' ❹
    }
    asciidoctorj {
        attributes 'pdf-themesdir': "@style",
                  'pdf-theme': 'pdf-theme.yml' ❺
    }
}
```

- ❶ 追加の Ruby スクリプトを最初にロードします。
- ❷ Asciidoc 文書の起点フォルダです。
- ❸ PDF 文書用のフォントフォルダ定義です。
- ❹ Asciidoc 文書の起点ファイルです。

⑤ PDF 文書のテーマ定義です。



asciidoctor 及び asciidoctorPdf タスクの実行結果出力先は、プロジェクトフォルダ下の build テンポラリーフォルダです。なお、ビルドを初期化する clean タスクの実態は build フォルダ内のファイルを削除する処理になっています。

また asciidoctorj 定義では、文書変換スクリプトで使われる AsciidoctorJ や周辺モジュールのバージョン設定をしています。

build.gradle の asciidoctorj 定義

```
asciidoctorj {  
    version = '2.5.6' ①  
    modules {  
        diagram.use()  
        diagram.version '2.2.8' ②  
        pdf.version '1.6.2' ③  
    }  
    requires = [ ④  
        'asciidoctor-diagram',  
        'prawn_svg_font_patch', ⑤  
        'asciidoctor/nabetani', ⑥  
    ]  
    attributes 'source-highlighter': 'rouge' ⑦  
}
```

- ① AsciidoctorJ のバージョン指定します。最新バージョンを指定できますが、破壊的変更がある場合にビルドが失敗する場合があるので、既存の文書のビルドでバージョンを上げる際には注意します。
- ② ダイアログ記法を処理する asciidoctor-diagram のバージョン指定です。最新バージョンが指定できます。こちらも既存文書でバージョンを上げる場合は、破壊的変更に注意します。
- ③ PDF 文書変換をする asciidoctor-pdf のバージョン指定です。日本語禁則処理が 1.6.2 に依存しているため本ビルドでは固定です。
- ④ Asciidoctor が使う Ruby スクリプトの定義です。追加のスクリプトがある場合は build.gradle ファイル上部の dependencies 部分で定義し gradle/repos/gem 配下に Ruby Gem として配置します。
- ⑤ ダイアログ記法で使われるフォントを日本語にパッチするスクリプトです。
- ⑥ PDF 文書で日本語禁則処理を行う asciidoctor-nabetani の導入定義です。asciidoctor-pdf 1.6.2 に依存します。

Asciidoc 文書変換で追加の処理が必要な場合は、Asciidoc カスタムコンバータを作成し Ruby Gem 形式でプロジェクトフォルダに配置しビルドに組み込んで実行できます。カスタムコンバータの詳細については以下のドキュメントを参照してください。



<https://docs.asciidoctor.org/asciidoctor/latest/convert/custom/>

Asciidoctor supports custom converters. If you want to produce an output format that's not supported by a built-in converter or any of the available converters in the ecosystem, you can create and use your own converter.

— Custom Converter | Asciidoctor Docs

起動タスクとなっている docs タスクでは、ここまでで解説した Asciidoctor Gradle Plugin が事前実行依存として設定しており、文書変換処理完了後に build フォルダに生成された文書及びリソースを docs フォルダにコピーします。

build.gradle の docs タスク定義

```
tasks.register('docs', Copy) {
    dependsOn(asciidoctor, asciidoctorPdf, cleanDocs)
    group 'documentation'
    description 'Generate HTML/PDF documents from Asciidoc documents and output them under docs/'
    from 'build/docs/asciidoc/index.html' ❶
    from 'build/docs/asciidocPdf/index.pdf' ❶
    from('build/docs/asciidoc/') { ❷
        include 'Chapter*/images/*'
    }
    into 'docs/'
}
```

- ❶ build に生成された PDF/HML 文書を docs にコピーします。
- ❷ HTML 文書用に build に出力された画像、ダイアログ記法で生成された SVG、表紙裏表紙カバー PDF を含む、各 images 配下の全ての画像リソースを docs にコピーします。

また、docs の依存にひとつとなっている cleanDocs タスクでは docs フォルダのクリーンアップを行っています。

build.gradle の cleanDocs タスク定義

```
tasks.register('cleanDocs', Delete) {
    group 'documentation'
    description 'Task to delete all files and directories under docs/'
    // docs/ ディレクトリは残しその配下のファイルとディレクトリを全て削除
    delete file('docs/').listFiles()
}
```



本項で現れたビルドスクリプト中に現れるパスを変更することで、プロジェクトフォルダのファイル・フォルダ構成を変更できます。

5.4. checkPdfInfo タスク

checkPdfInfo タスクは docs/index.pdf に生成された PDF のメタ情報を出力します。

手順. checkPdfInfo タスクの実行

```
./gradlew -q checkPdfInfo
```



gradlew の -q` オプションは Gradle からのビルドメッセージ出力を抑止し、タスクからの出力を見やすくするオプションです。

このタスクの出力結果は次のようになります。

手順. checkPdfInfo タスクの出力例

```

Content-Type: application/pdf
access_permission:assemble_document: true
access_permission:can_modify: true
access_permission:can_print: true
access_permission:can_print_degraded: true
access_permission:extract_content: true
access_permission:extract_for_accessibility: true
access_permission:fill_in_form: true
access_permission:modify_annotations: true
dc:creator: https://github.com/h1romas4/asciidoctor-gradle-template
dc:format: application/pdf; version=1.4
dc:title: AsciidoctorとGradleでつくる文書執筆環境
dcterms:created: 2023-10-04T06:17:20Z
dcterms:modified: 2023-09-25T08:40:46Z
pdf:PDFVersion: 1.4
pdf:annotationSubtypes: Link
pdf:annotationTypes: null
pdf:charsPerPage: 0
pdf:containsDamagedFont: false
pdf:containsNonEmbeddedFont: false
pdf:docinfo:created: 2023-10-04T06:17:20Z
pdf:docinfo:creator: https://github.com/h1romas4/asciidoctor-gradle-template
pdf:docinfo:creator_tool: Asciidoctor PDF 1.6.2, based on Prawn 2.4.0
pdf:docinfo:modified: 2023-09-25T08:40:46Z
pdf:docinfo:producer: https://github.com/h1romas4/asciidoctor-gradle-template
pdf:docinfo:title: AsciidoctorとGradleでつくる文書執筆環境
pdf:encrypted: false
pdf:hasCollection: false
pdf:hasMarkedContent: false
pdf:hasXFA: false
pdf:hasXMP: false
pdf:num3DAnnotations: 0
pdf:overallPercentageUnmappedUnicodeChars: 0.0
pdf:producer: https://github.com/h1romas4/asciidoctor-gradle-template
pdf:totalUnmappedUnicodeChars: 0
pdf:unmappedUnicodeCharsPerPage: 0
xmp:CreatorTool: Asciidoctor PDF 1.6.2, based on Prawn 2.4.0
xmpTPg:NPages: 77

```

PDF 文書のメタ情報には、主に `src/docs/asciidoc/index.adoc` で設定したタイトルや著者名などが記録されますが、本タスクで正しく設定されているかを確認できます。

5.5. checkSyncImage タスク

`checkSyncImage` タスクは、Asciidoc 文書から使われている画像リソースと `src/docs/asciidoc` 内のファイルを比較し一致性を出力します。特に未使用画像を検出するのに便利なタスクです。

手順. checkSyncImage タスクの実行

```
./gradlew -q checkSyncImage
```

このタスクの出力結果は次のようになります。

手順. checkPdfInfo タスクの出力例

```

::Unused images:
Chapter01/images/unused-image-1.png
Chapter03/images/unused-image-1.png
::Image not found:
Chapter01/images/2023-09-17-13-43-06.png

```

それぞれの出力項目の内容は次のとおりです。なお、タスクの実行で何も出力されず正常終了した場合は、Asciidoc 文書と画像ファイルが整合していることを示します。

::Unused images:

Asciidoc 文書からリンクされていない未使用画像ファイルの一覧です。これらは不要なファイルとして削除できます。

::Image not found:

Asciidoc 文書からリンクされているが存在しない画像ファイルの一覧です。画像リンク切れファイルはビルド時の docs タスクでも警告となり、文書を修正する必要があります。



::Unused images: に申告されたファイルは未使用ながら docs 配下の HTML 文書用画像フォルダに転送されます。HTML 文書の配布を行う場合は、本タスクで検出して src/docs/asciidoc フォルダから削除後、再度文書のビルドをしてクリーンアップすると良いでしょう。

本タスクの設定は、build.gradle の次の記述で行われています。

build.gradle の checkSyncImage タスク定義

```

tasks.register('checkSyncImage', SyncImageTask) {
    group 'documentation'
    description 'Utility to check the consistency of Asciidoc text and file system images'
    // 起点の文書ディレクトリ
    baseDir = file('src/docs/asciidoc')
    // 起点の Asciidoc 文書 (include を辿る)
    index = 'index.adoc'
    // ファイルシステム上で画像として認識する拡張子
    // 文書内で image:: として使われているファイル全ての拡張子を設定すること
    imageExt = ['png', 'jpg', 'jpeg', 'svg']
}

```

画像ファイルは imageExt 変数で設定された拡張子をもって識別しています。

もし設定されている以外の画像拡張子が Asciidoc 文書で使われている場合は、::Image not found: としてリンク切れで認識されます。必要に応じて設定を修正してください。

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

2025年 2月 27日 v4.2.0 発行

著者	田中 広将
カバーデザイン	Fujix
ライセンス	MIT License
発行	https://github.com/h1romas4/asciidoctor-gradle-template
連絡先	https://github.com/h1romas4/asciidoctor-gradle-template/issues

