

# AsciiDoc ユーザーマニュアル

K1.0.0, 2019/10/21

# 目次

1. 配布パッケージについて.....	1
2. AsciiDocについて .....	2
3. 仕様書作成にAsciiDocを選択した背景 .....	3
3.1. 従来の仕様書作成の問題点 .....	3
3.2. AsciiDocによる仕様書作成のメリット.....	4
4. 環境構築手順 .....	5
4.1. はじめに.....	5
4.2. Chocolateyをインストール.....	6
4.3. Chocolateyのリポジトリから各種パッケージをインストール.....	7
4.4. AsciiDoc関連ツールをインストール .....	13
4.5. VScodeの拡張機能をインストール .....	14
5. 使い方.....	16
5.1. VScodeで実際にAsciiDocを書いてみる.....	16
5.2. AsciiDocで書いた文章をGitで管理する .....	22

# 1. 配布パッケージについて

- 中身は以下の通り

```
AsciiDocPackage/  
└ template/                                // 文章のテンプレート一式  
  └ csv/                                    // CSVファイルを格納  
  └ fonts/                                // フォントファイルを格納  
  └ images/                                // イメージファイルを格納  
  └ style/                                // スタイルファイルを格納  
  └ .gitignore                             // コミット対象外の指定用  
  └ make_pdf_test.bat                     // AsciiDoc→PDF変換用スクリプト  
  └ test.adoc                             // テストサンプル  
  └ test2.adoc                            // テストサンプルその2  
└ tool/                                    // インストール用バッチファイル等  
  └ ①ChocolateyInstall.bat  
  └ ②ChocolateyPackageInstall.bat  
  └ ③AsciiDocToolInstall.bat  
  └ ④VScodeExtensionInstall.bat  
  └ ndp48-web.exe  
└ AsciiDocUserManual.pdf                 // 手順書
```

## 2. AsciiDocについて

### • AsciiDocとは？

- 文章を記述するための記法(軽量マークアップ言語)の一つ
- プレーンテキストで体裁が整った文章が書ける
- そのままでも可読性が高い(AsciiDoc ≒ Markdown > HTML)
- Markdownよりも表現力が高い(HTML > AsciiDoc > Markdown)



公式サイト

<http://asciidoc.org/>

## 3. 仕様書作成にAsciiDocを選択した背景

### 3.1. 従来の仕様書作成の問題点

- **不適切なバージョン管理**
  - ガバガバな命名規則
    - 日付ファイル名(作成日?編集日?発行日?)
    - 編集者名や編集中や最新と付いたファイル
  - 管理場所が複数
    - バックアップ用にコピーファイルが散乱
    - Originalファイルの修正に追従しづらい
- **不適切な履歴管理**
  - 更新履歴シート
    - 複数の変更箇所もまとめて更新しがち
    - 変更の経緯までは残しにくい
    - そもそも書かない
  - 変更箇所を探しにくい
    - 変更箇所は色付き(次の編集者が色消して追えなくなる)
    - 簡単に差分抽出ができない
- **不適切な品質管理**
  - レビューされない
    - 間違った仕様や完成度の低い仕様がそのままリリースされる
    - 変更内容が共有されない(担当者のみぞ知る)
- **本質的ではない作業**
  - 罫線、文字色、フォント種別、目次、章番号
    - いちいちキーボードからマウスに持ちかえてリボンから選択
    - 選択肢が多くて無駄に選ぶ時間がかかる
- **生産性**
  - とにかく重くなりがち
    - ファイルサイズの肥大化により開けない、スクロールが遅い
    - Microsoft Wordは動作を停止しました…

## 3.2. AsciiDocによる仕様書作成のメリット

- 不適切なバージョン管理、履歴管理、品質管理

- バージョン管理ツールのGitとの相性が良い
  - 仕様書を一元管理できる
  - Originalファイルへは影響を与えず(常にリリース可能な状態に保たれる)ローカルで編集可能
  - 必然とローカル環境に複製されるので分散開発しやすく障害に強い
  - 変更は全て記録されていて、過去の変更を簡単に参照できる
  - テキストベースなので変更箇所の差分抽出が容易にできる
  - プルリクエストによりメンバーに周知とレビューを兼ねられる

- 本質的ではない作業、生産性

- AsciiDocが解決!
  - 煩わしいマウス操作は不要で全てテキストベースで作業が行える(文章構造の明示や装飾、テーブル記法まで)
  - 記法が少ないことで良い意味で制限がかかり、担当者差が出にくくドキュメントに統一感が出る
  - 編集するツールに限定されない(書くだけならエディタは何でもよい)
  - テキストそのままで可読性の高いドキュメントになるため必然的に簡潔な内容になりレビューしやすい
  - 対応アプリの拡張機能で簡単にプレビュー環境をつくれて快適に読み書きできる
  - シーケンス図などをPlantUMLでテキストベースで書いて埋め込み可能
  - 外部ファイルのインクルードも可能
  - コードのコメントアウトが可能(可読性は保ちつつ、変更の経緯や設計根拠も残しやすい)
  - 展開用にHTML化やPDF化なども可能
  - テキストベースなのでとにかく軽い!

## 4. 環境構築手順

### 4.1. はじめに

本書では、AsciiDocのテキストエディタとして Visual Studio Code を利用することとします。  
また、AsciiDocドキュメントのバージョン管理は、Gitを視覚的に操作可能なSourceTreeを使います。

以下の環境で動作を確認しています。

- Windows 10 Home (64bit)
- .NET Framework 4.0以上
- Chocolatey 0.10.15
  - ruby 2.6.3.1
    - asciidoctor 2.0.10
    - asciidoctor-pdf 1.5.0.beta.3
    - asciidoctor-pdf-cjk 0.1.3
    - asciidoctor-diagram 1.5.18
    - coderay 1.1.2
  - Graphviz 2.38.0.20190211
  - jdk8 8.0.221
  - Maven 3.6.1.20190711
  - Visual Studio Code 1.38.1
    - AsciiDoc 2.7.6
    - Japanese Language Pack for Visual Studio Code 1.37.5
    - PlantUML 2.12.1
  - Winmerge 2.16.4.20191007
  - SourceTree 3.1.3

これらのツールを自動でインストール(一部除く)するためのバッチファイルを用意しています。  
コマンドプロンプト上で長時間のバッチ処理を行うにあたり、誤って処理を止めてしまわないように、事前に以下の設定を行ってください。

1. Windowsのスタートメニューから **コマンドプロンプト** を検索して起動
2. システムメニュー(コマンドプロンプトウィンドウの左上に表示されているアイコンをクリックすると表示されるメニュー)から **プロパティ** を選択
3. 編集オプション内の **簡易編集モード** のチェックを外して、**OK** をクリック
4. コマンドプロンプトを閉じる

## 4.2. Chocolateyをインストール

### 実施手順

1. 以下のバッチファイルをダブルクリックで実行

①ChocolateyInstall.bat

2. ユーザーアカウント制御の許可のポップアップが出るので **はい** をクリック
3. コマンドプロンプトが表示されて処理が進むので自動的に閉じたら完了(1分程度かかります)

### 実行内容の覚え書き(実施は不要)

- コマンドプロンプト(管理者権限)で以下を実行している

```
@"%SystemRoot%\System32\WindowsPowerShell\v1.0\powershell.exe" -NoProfile  
-InputFormat None -ExecutionPolicy Bypass -Command "iex ((New-Object  
System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))"  
&& SET "PATH=%PATH%;%ALLUSERSPROFILE%\chocolatey\bin"
```



公式サイト

<https://chocolatey.org/install#installing-chocolatey>

インストール手順解説(日本語)

<https://qiita.com/konta220/items/95b40b4647a737cb51aa>

### Chocolateyとは?

- Windows上で動作するソフトウェアをコマンドラインでパッケージ管理可能なツール

### メリット

- Chocolateyのリポジトリに登録されているパッケージを**一発でインストール**できる
- Chocolateyでインストールしたソフトは**一括でアップデート**できる



## 4.3. Chocolateyのリポジトリから各種パッケージをインストール

### 実施手順

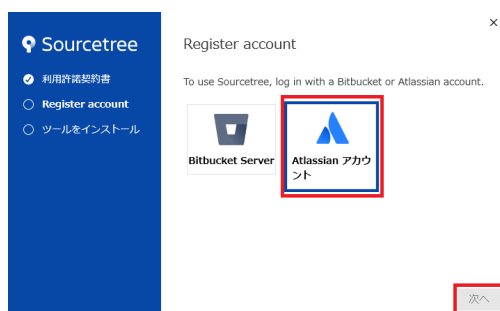
1. 以下のバッチファイルをダブルクリックで実行

②ChocolateyPackageInstall.bat

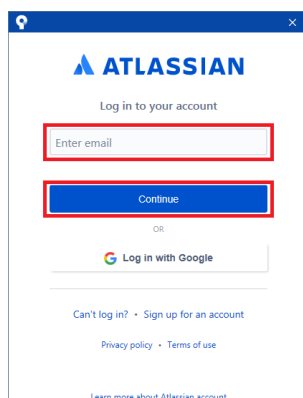
2. ユーザーアカウント制御の許可のポップアップが出るので **はい** をクリック
3. コマンドプロンプトの画面内にて、OSのバージョンがWindows10かどうか聞かれるので、Windows10であれば **y**、Windows7であれば **n** を入力して **Enter** を押す  
**y** の場合：次の手順へ  
**n** の場合：コマンドプロンプトの画面内に書いてある手順に従う
4. コマンドプロンプトが表示されて処理が進むのでしばらく待つ(20分程度かかります)
5. 下記画面が表示されたら **ライセンスに同意します** にチェックを入れて **次へ** をクリック



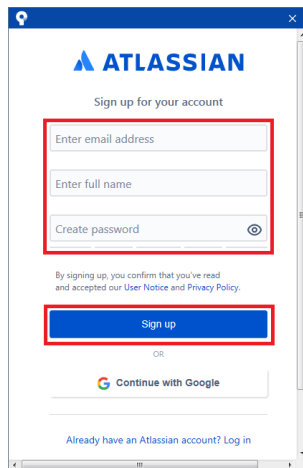
6. Atlassianアカウントを選択して **次へ** をクリック



7. メールアドレスを入力して **Continue** をクリック



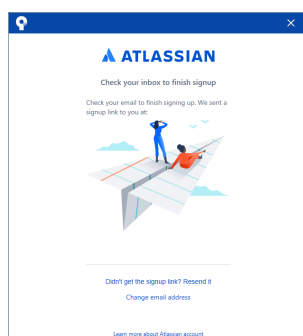
8. 任意のユーザー名とパスワードを入力して、**Sign up** をクリックで、アカウントを作成

A screenshot of the Atlassian sign-up page. The page has a blue header with the Atlassian logo. Below the logo, it says "Sign up for your account". There are three input fields: "Enter email address", "Enter full name", and "Create password" (with an eye icon for toggling visibility). Below these fields is a blue "Sign up" button. Under the button, it says "OR" and "Continue with Google" with the Google logo. At the bottom, it says "Already have an Atlassian account? Log in".

9. reCAPTCHAの画像認証の指示に従って選択を行い、**確認** をクリック



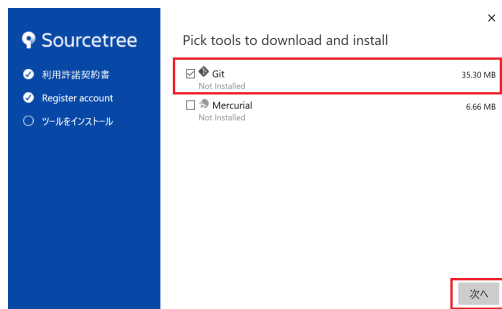
10. メールアドレスの認証メールが届くので、認証ボタンをクリック後、下の画面を閉じる



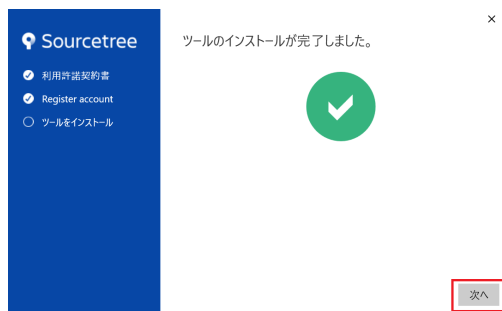
11. 再度、手順7の画面になり、先ほど作成したアカウントでログインすると、登録完了画面に遷移するので、**次へ** をクリック



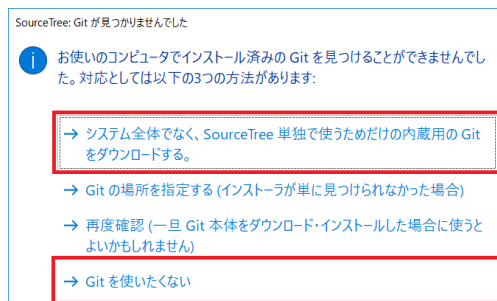
12. ツールのインストール画面に遷移するので、Git にのみチェックを入れて、次へ をクリック



13. Gitのダウンロードが完了したら、次へ をクリック



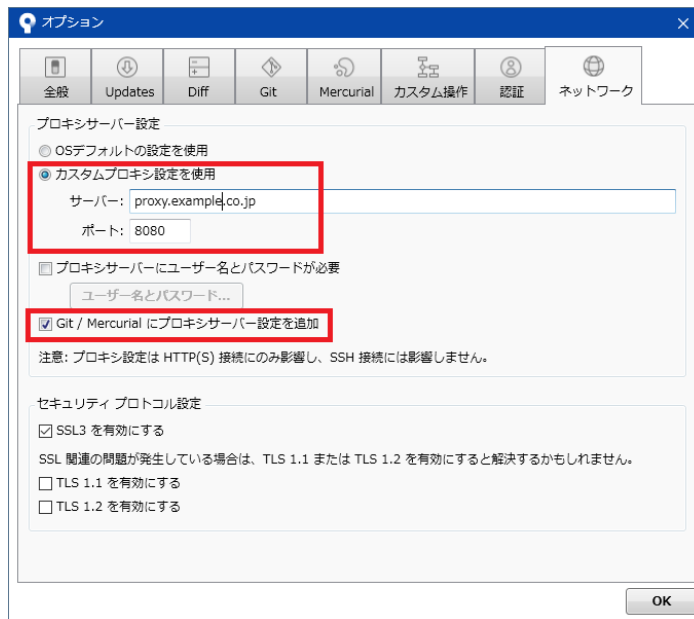
- 社内のProxy環境化の場合、ダウンロードに失敗してエラー画面になるので、一度インストール画面を閉じる
- Windowsのスタートメニューから Sourcetree を検索して起動
- 下記画面が表示されたら、一番下の選択肢の Gitを使いたくない をクリック



- SourceTreeが起動するので、[ツール]→[オプション]を開く



- e. ネットワーク タブを選択し、カスタムプロキシ設定を使用 にチェックを入れてProxyを設定し(サーバーの頭にhttp://は不要)、Git/Mercurialにプロキシ・・・にチェックを入れる



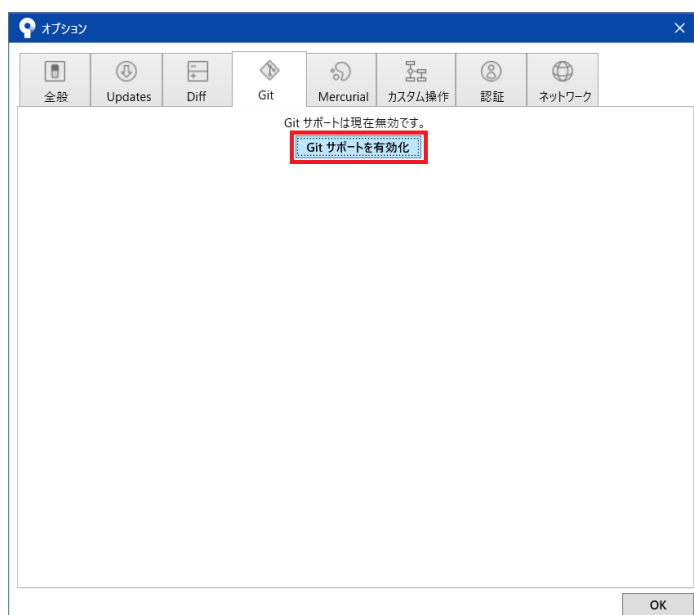
proxyの確認方法 (Windows10の場合)

<https://pasokatu.hateblo.jp/entry/2017/07/04/111147>

proxyの確認方法 (Windows7の場合)

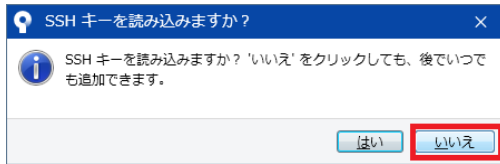
<https://pc-karuma.net/internet-explorer-proxy-settings/>

- f. 続けて、Git タブを選択し、Gitサポートを有効化 をクリック



- g. 再度、手順Cの画面が表示されるので、一番上の選択肢の **システム全体でなく、・・・** をクリック
- h. Gitのダウンロードが始まるので、完了したら **OK** をクリックして、Sourcetreeを閉じる(手順17へ)

14. **SSHキーを読み込みますか？** が表示されたら **いいえ** をクリック



15. Sourcetreeが起動したら、一旦閉じる
16. コマンドプロンプトの画面内にて、Atlassianアカウントを作成完了したか聞かれるので、少なくとも手順11まで完了していれば **y**、完了していなければ **n** を入力して **Enter** を押す  
**y** の場合：コマンドプロンプトが表示されて処理が進むので自動的に閉じたら完了(5分程度かかります)  
**n** の場合：コマンドプロンプトの画面内に書いてある手順に従う(再度、**Enter** を押すと画面が閉じる)

※上記処理が完了後、Sourcetreeを起動すると初回のみ起動前にいくつか画面が表示されるが、気にせず全て **次へ** をクリックする

## 実行内容の覚え書き(実施は不要)

- コマンドプロンプト(管理者権限)で以下を実行している

```
cinst ruby -y ①
cinst graphviz -y ②
cinst jdk8 -y ③
cinst maven -y ④
cinst vscode -y ⑤
cinst winmerge -y ⑥
cinst sourcetree --version 2.5.5 -y ⑦
```

- ① Ruby (AsciiDoc関連ツールを利用するのに必要)
- ② Graphviz (PlantUMLのレンダリングライブラリとして必要)
- ③ Java (PlantUMLの動作環境として必要)
- ④ Maven (Javaのプロジェクト管理ツールでPlantUMLサーバー立ち上げに必要)
- ⑤ Visual Studio Code (AsciiDocをプレビュー可能なテキストエディタ)
- ⑥ Winmerge (コードの差分比較ツール)
- ⑦ SourceTree (GitのGUIツール)

- Atlassianアカウントを作成してSourceTreeのサインインに成功したら、コマンドプロンプト(管理者権限)で以下を実行しSourceTreeのアップデートを行う

```
choco upgrade all -y
```



初めから最新verをインストールしないのはBitbucketに登録せずに利用するため  
<https://hepokon365.hatenablog.com/entry/2019/03/25/222814>

## 4.4. AsciiDoc関連ツールをインストール

### 実施手順

1. 以下のバッチファイルをダブルクリックで実行

```
③AsciiDocToolInstall.bat
```

2. コマンドプロンプトが表示されて処理が進むので自動的に閉じたら完了(2分程度かかります)

### 実行内容の覚え書き(実施は不要)

- コマンドプロンプトで以下を実行している

```
gem install asciidoctor ①  
gem install --pre asciidoctor-pdf ②  
gem install asciidoctor-pdf-cjk ③  
gem install asciidoctor-diagram ④  
gem install coderay ⑤
```

① AsciiDoc→HTMLに変換用

② AsciiDoc→PDFに変換用

③ PDF変換のレイアウト崩れ対応用

④ PlantUML等の図の記述および出力用

⑤ コードのシンタックスハイライト用

- 社内のProxy環境化で実行する場合はgemにproxyを指定

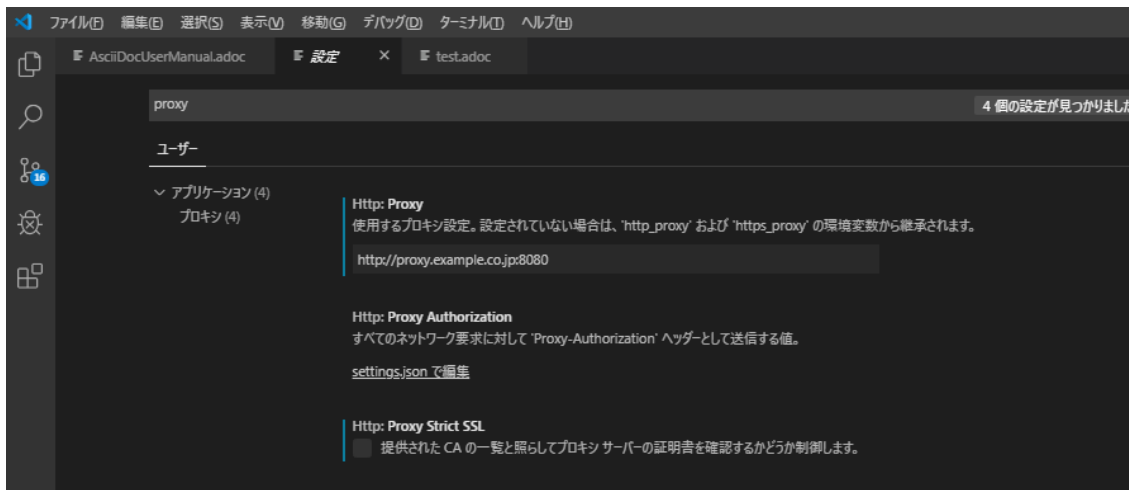
```
gem install xxxx -p http://アドレス:ポート
```

## 4.5. VScodeの拡張機能をインストール

### 実施手順

1. Windowsのスタートメニューから **Visual Studio Code** (以下、VScodeとする) を検索して起動
2. ショートカット `Ctrl+,` で設定を開き、`http: proxy` を検索して、以下を設定

```
"http.proxy": "http://アドレス:ポート",      //Proxyの確認手順は先述の通り  
"http.proxyStrictSSL": false,                //チェックを外す
```



3. 起動中のVScodeを閉じる
4. 以下のバッチファイルをダブルクリックで実行

```
④VScodeExtensionInstall.bat
```

5. コマンドプロンプトが表示されて処理が進むので自動的に閉じたら完了(1分程度かかります)
6. 手順2の設定を元に戻す

### 実行内容の覚え書き(実施は不要)

- コマンドプロンプトで以下を実行している

```
code --install-extension joaompinto.asciidoctor-vscode ^ ①  
code --install-extension MS-CEINTL.vscode-language-pack-ja ^ ②  
code --install-extension jebbs.plantuml ^ ③
```

- ① VScodeのASciiDocプラグイン(プレビュー用)
- ② VScodeの日本語対応プラグイン
- ③ VScodeのPlantUMLプラグイン(UML図の爆速プレビュー用)





[表示]→[拡張機能]から検索してインストール or コマンドラインからインストール  
<https://qiita.com/Kosen-amai/items/03632dee2e1694652f06>

VScodeにProxyを設定する方法

<https://qiita.com/cointoss1973/items/b3c84daeed90fd183501>

※最後にProxyの設定を元に戻すのは、PlantUMLで爆速プレビューを行うためのおまじない

## 5. 使い方

### 5.1. VScodeで実際にAsciiDocを書いてみる

ここでは、テストサンプルのプレビューを行い、正しく環境構築ができたことを確認します。  
テストサンプルの内容は、AsciiDocの文法紹介も兼ねているので参考にしてください。

#### 作業ディレクトリを作成する

配布パッケージ内のtemplateフォルダー式をローカルPCの任意の場所にコピーして使います。  
このフォルダー式が文章のテンプレートとなります。

#### 作業ディレクトリ作成の覚え書き(実施は不要)

- 文章作成のための作業ディレクトリを用意

```
└ template/                // 文章のテンプレート一式
├ csv/                     // CSVファイルを格納
├ fonts/                   // フォントファイルを格納
├ images/                  // イメージファイルを格納
└ style/                   // スタイルファイルを格納
```

- HTMLのスタイルファイルを用意  
asciidocの配布ファイルがWindowsの場合は以下にあるのでコピーして利用

```
// ruby2.6でasciidocのverが2.0.10の場合
C:\tools\ruby26\lib\ruby\gems\2.6.0\gems\asciidoc-
2.0.10\data\stylesheets\asciidoc-default.css
```

- PDFのスタイルファイルを用意  
asciidoc-pdfの配布ファイルがWindowsの場合は以下にあるのでコピーして利用

```
// ruby2.6でasciidoc-pdfのverが1.5.0.beta.2の場合
C:\tools\ruby26\lib\ruby\gems\2.6.0\gems\asciidoc-pdf-
1.5.0.beta.2\data\themes\default-theme.yml
```



デフォルトのスタイルを適用するだけなら、`:stylesdir:` をアトリビュートに指定するだけでOK  
今回展開するtemplateではメタ情報の埋め込みやレイアウト修正を施したスタイルを使用します

公式サイト

<https://github.com/asciidoc/asciidoc-pdf/blob/master/docs/theming-guide.adoc>

- フォントファイルを用意  
asciidoc-pdfの配布ファイルがWindowsの場合は以下にあるのでコピーして利用

```
// ruby2.6でasciidoctor-pdfのverが1.5.0.beta.2の場合  
C:\tools\ruby26\lib\ruby\gems\2.6.0\gems\asciidoctor-pdf-  
1.5.0.beta.2\data\fonts\*.ttf
```



カスタマイズ参考サイト

<https://ryuta46.com/267>

<https://qiita.com/kuboaki/items/67774c5ebd41467b83e2>

- ドキュメントファイルを用意  
適当にメモ帳で以下の設定で作成する

```
拡張子 : .adoc  
文字コード : UTF-8
```

- 格納後の作業フォルダ内はこんな感じになる

```
├ template/  
│   ├── csv/  
│   ├── fonts/  
│   │   ├── *.ttf  
│   │   └ ...  
│   ├── images/  
│   └ style/  
│       ├── asciidoctor-default.css  
│       ├── default-theme.yml  
│       └ public_style.yml  
└ *.adoc
```

## VScode を起動する

AsciiDocで書くためのテキストエディタとして使用します。

Windowsのスタートメニューから **Visual Studio Code** (以下、VScodeとする)を検索して起動します。

## テストサンプルを開く

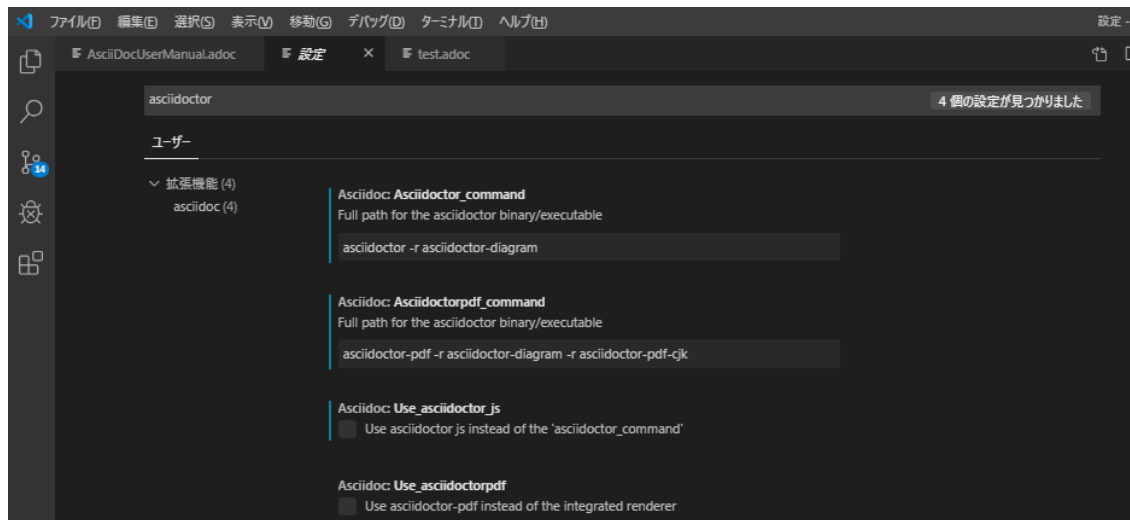
[ファイル]→[ファイルを開く]から **template** フォルダ内の **test.adoc** ファイルを開きます。

## テストサンプルをプレビューする

### asciidoctorの設定を変更する(初回のみ実施)

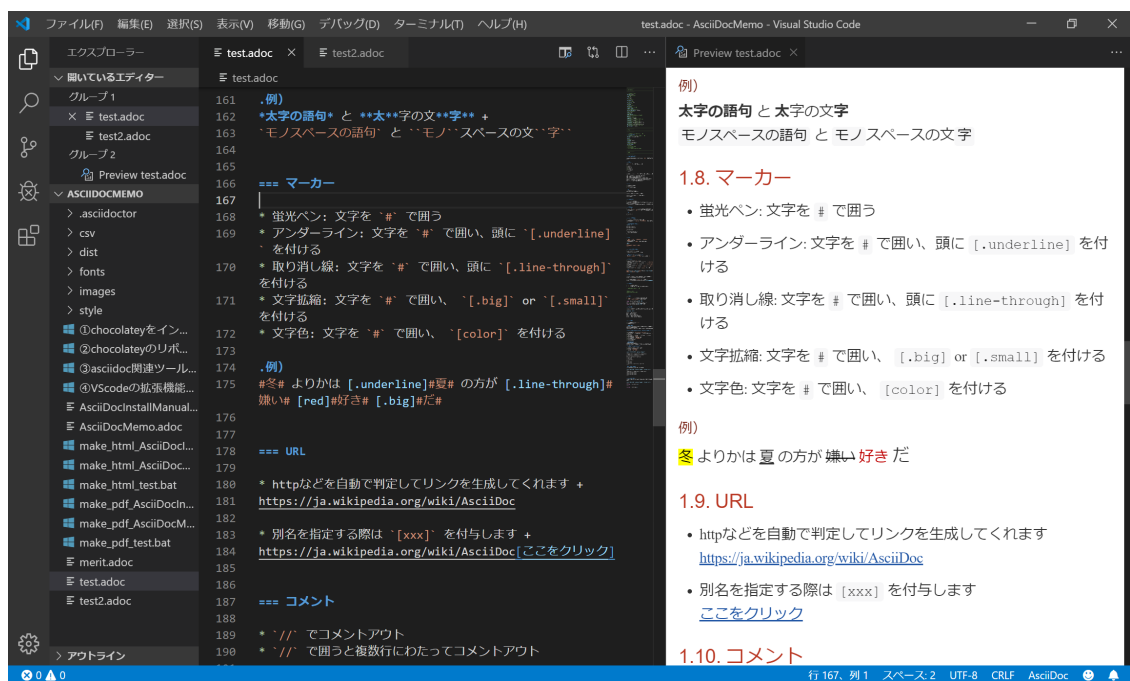
- ショートカット **Ctrl+,** で設定を開き、**asciidoctor** を検索し、以下を設定

```
"AsciiDoc.asciidoc_command": "asciidoc -r asciidoc-diagram",
"AsciiDoc.asciidoc_command": "asciidoc-pdf -r asciidoc-diagram -r
asciidoc-pdf-cjk",
"AsciiDoc.use_asciidoc_js": false (チェックを外す),
```



## プレビューを行う

- カーソルで選択中にショートカット `Ctrl+K → V` で画面右側にプレビューが表示



## プレビューを行う(UML図の編集時)

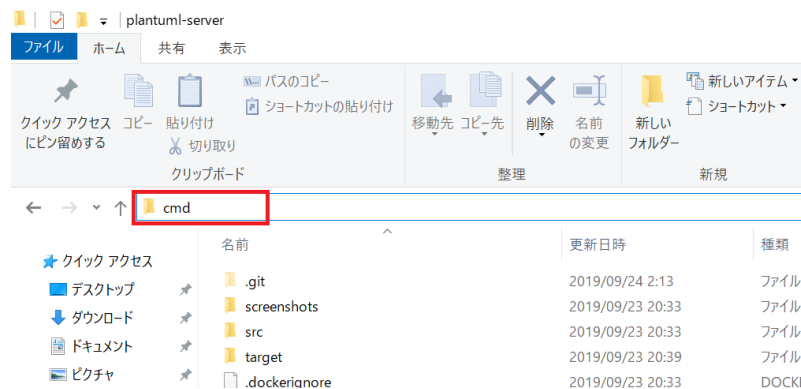
※この手順は参考情報なので読み飛ばしてもOK (UML編集時に直面する問題に対する対応)

問題点:

- AsciiDocプラグインにてUML図をプレビューすると、編集してプレビューする度に画像ファイルが生成される
- UML図にファイル名を指定することで上書き編集となり画像の増殖を防げるが、今度はキャッシュが効いてリアルタイムな編集が難しい

#### 解決策:

- プレビュー用のPlantUMLサーバーをローカルに立ち上げ、PlantUMLプラグインを使ってプレビューを行う
  1. 以下から、PlantUML Server リポジトリをローカルPCの適当な場所にクローン（初回のみ実施）  
<https://github.com/plantuml/plantuml-server>
  2. コマンドでサーバーを立ち上げる
    - a. 手順1にてクローンした場所をエクスプローラーで開く
    - b. 上記エクスプローラーのアドレスバーに `cmd` と入力して `Enter` を押し、コマンドプロンプトでこの場所を開く



起動したコマンドプロンプトにて、下記コマンドを打ち、しばらくするとサーバーが立ち上がる(1分程かかります)

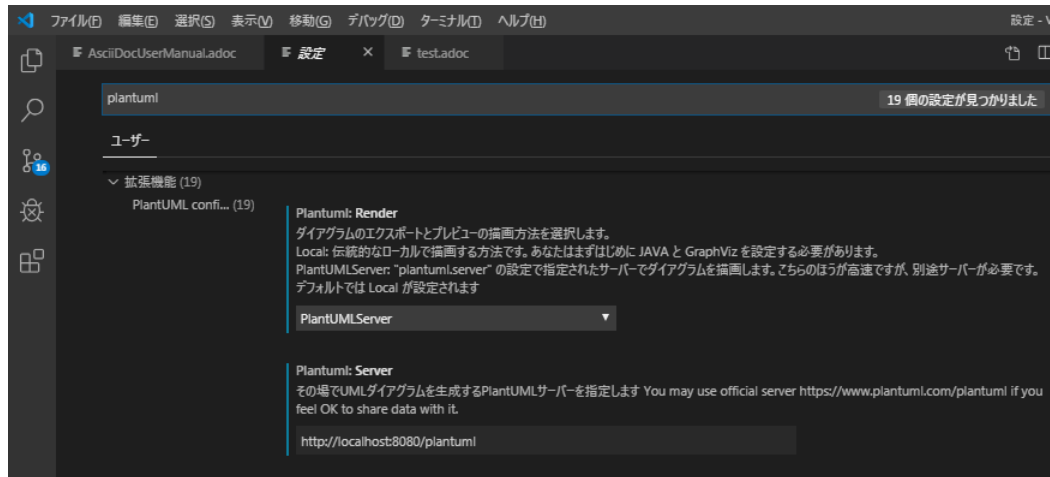
```
mvn jetty:run
```

成功すると、コマンドの最後に下記が表示される

```
[INFO] Starting scanner at interval of 5 seconds.
```

3. VScodeにて、ショートカット `Ctrl+,` で設定を開き、`plantuml` を検索し、以下を設定（初回のみ実施）

```
"plantuml.render": "PlantUMLServer",  
"plantuml.server": "http://localhost:8080/plantuml",
```



#### 4. UML図の編集

- a. UMLのブロック内の最初と最後に `@startuml` と `@enduml` を指定

```
[plantuml, "画像ファイル名"]
--
@startuml

@enduml
--
```

- b. カーソルで選択中にショートカット `Alt+D` で画面右側にプレビューが表示



- 編集が終わったら、`@startuml` と `@enduml` の指定を消して、AsciiDocプラグイン側でプレビューすれば画像ファイルを上書き可能
- PlantUML爆速プレビュー  
<https://qiita.com/Ping/items/64930e8c21fb95bec095>
- PlantUML図の書き方  
<https://qiita.com/ogomr/items/0b5c4de7f38fd1482a48>  
<http://yohshiy.blog.fc2.com/blog-category-22.html>

## テストサンプルをPDFに変換する

1. 以下のバッチファイルをダブルクリックで実行

```
make_pdf_test.bat
```

2. コマンドプロンプトが表示されて処理が進むので自動的に閉じたら完了 (数秒程度かかります)  
3. 同じ階層に `test.pdf` が生成される



必要に応じて適当にメモ帳で開いてバッチファイル内のファイル名を修正して使ってください  
-o 変換後ファイル名.pdf 変換前ファイル名.adoc

## HTMLやPDFへの変換方法の覚え書き(実施は不要)

- コマンドプロンプトで以下を実行している(\*にファイル名を指定)

```
chcp 65001 ①  
asciidoctor -r asciidoctor-diagram -o *.html *.adoc ②  
asciidoctor-pdf -r asciidoctor-diagram -r asciidoctor-pdf-cjk -o *.pdf  
*.adoc ③
```

- ① コマンドプロンプトで使用する文字コードをUTF-8に変更
- ② AsciiDoc→HTML化用コマンド
- ③ AsciiDoc→PDF化用コマンド



文字コードの設定

<https://www.adminweb.jp/command/display/index5.html>

※AsciiDocはUTF-8を使用する必要があるが、WindowsのデフォルトがWindows-31Jのため変更

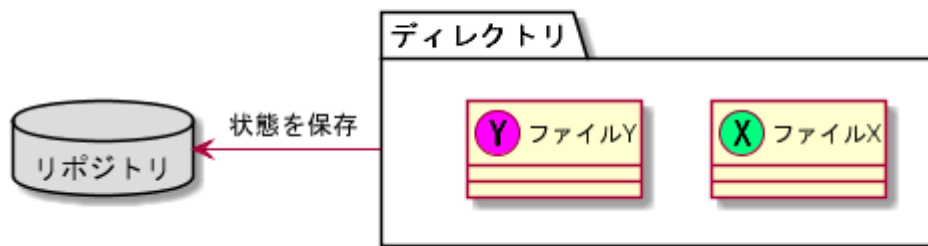
## 5.2. AsciiDocで書いた文章をGitで管理する

### 5.2.1. Gitとは？

分散型バージョン管理システムの一つです。

**リポジトリ**と呼ばれる記録場所に、管理したい**ファイル**や**ディレクトリ**の**状態を記録(変更履歴を保存)**することができます。

リポジトリを複数用意できる(開発者がそれぞれのローカルにリポジトリを持てる)ので、分散型と呼ばれています。



以降では簡単な説明にとどめますので、以下のサイトも参考にして下さい。

<サルでもわかるGit入門>

<https://backlog.com/ja/git-tutorial/>

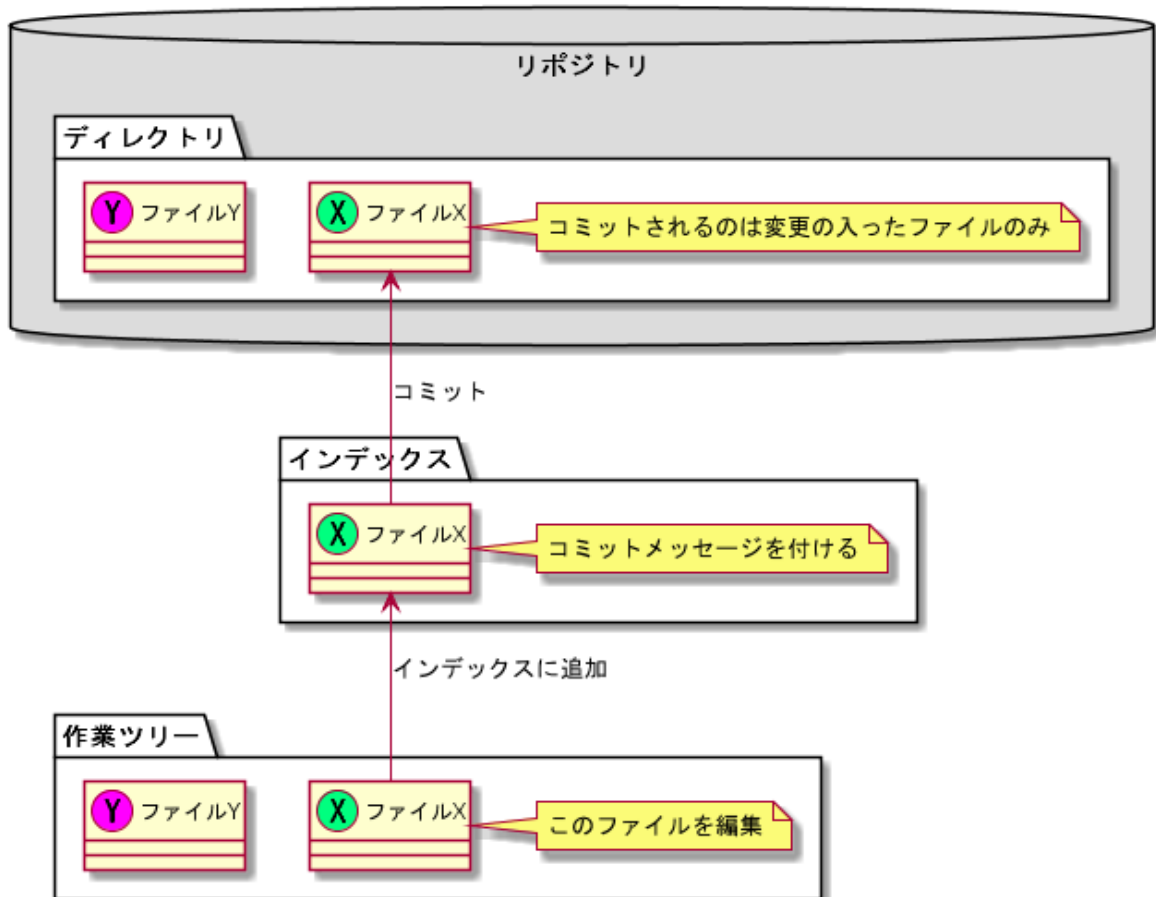


## 5.2.2. 状態の変化を記録するコミット

**コミット**と呼ばれる操作により、変更作業により発生したファイルやディレクトリの状態の変化をリポジトリに記録します。

1. 管理下に置かれた作業ディレクトリ(**作業ツリー**)から、コミット予定エリア(**インデックス**)に、変更したファイルを**追加(ステージング)**します。
2. **コミットメッセージ**を付けることができ、**コミット**を実行すると、前回コミットした時の状態から現在の状態までの差分を記録したコミットが作成されます。

```
//コミットメッセージ
1行目 : コミットでの変更内容の要約
2行目 : 空行
3行目以降 : 変更した理由
```



- このようにインデックスを挟むことで、変更されていないファイルを含めずにコミットでき、また、ファイルの一部の変更だけを追加してコミットすることもできたりします。

- コミットするとリポジトリで管理されている対象の全てのファイルが記録されていきます。しかしながら、ビルドのたびに更新される大量のファイル (例えばPlantUMLはプレビューするだけで画像が「.asciidoc」フォルダに生成され増殖していく) は、バージョン管理をしなくても問題ないものがほとんどです。

それらをコミット対象外とするのに「.gitignore」ファイルでの指定を行うことをお勧めします (一度プッシュ済みのファイルは後からは除外できなくなるので注意してください)。

<https://qiita.com/hainare/items/81435ac6912f2976cc9f>

過去のコミットを辿れば、過去の状態に一時的に戻したり参照したりすることができます。  
そのため、バックアップ用のファイルを別名で残す必要はありません。

### 5.2.3. 履歴を管理するリポジトリ

リポジトリには2種類あります。

- ・ リモートリポジトリ

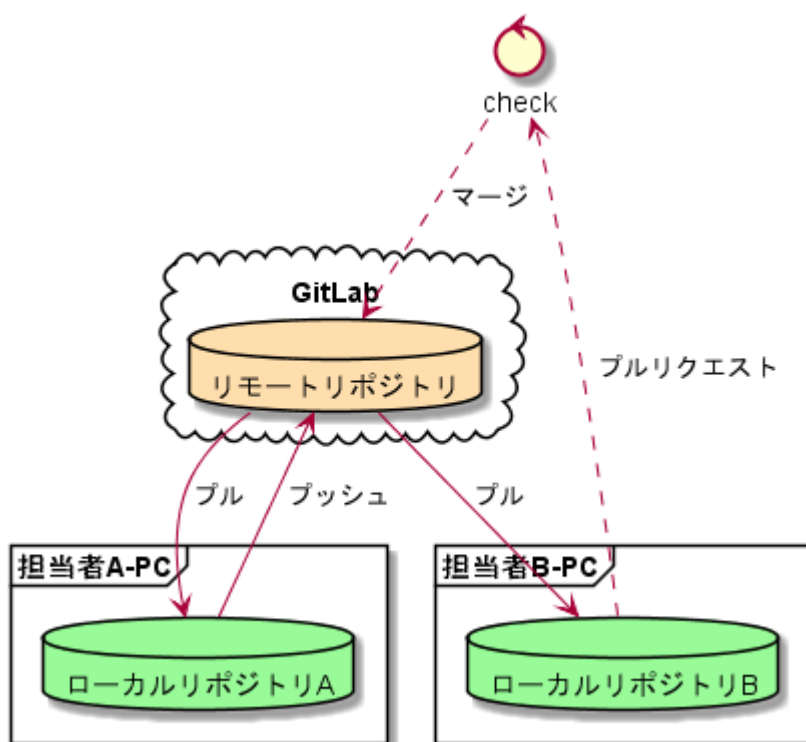
- 専用のサーバに配置して複数人で共有するためのリポジトリ

- ・ ローカルリポジトリ

- ユーザー一人ひとりが利用するために、自分の手元のPC上に配置するリポジトリ

普段の作業はローカルリポジトリで行い、作業が完了したらリモートリポジトリにアップロード(**プッシュ**)して公開します。

リモートリポジトリを通して公開された最新のディレクトリの状態を別の開発者が取得(**プル**)することもできます。



リモートリポジトリには、無料で使えるがインターネット上に一般公開されるGitHubと、自前のサーバに立ててプライベートリポジトリとして使えるGitLabがある(自社では後者を使用する)

なお、プッシュの代わりに、**プルリクエスト**を使うことで組織にレビューの文化を根付かせることも可能です。

プルリクエストは次のような機能を提供します。

- ・ 機能追加や改修など、作業内容がレビュー・マージ担当者やその他関係者に通知される
- ・ ソースコードの変更箇所がわかりやすく表示される
- ・ ソースコードに関するコミュニケーションの場が提供される

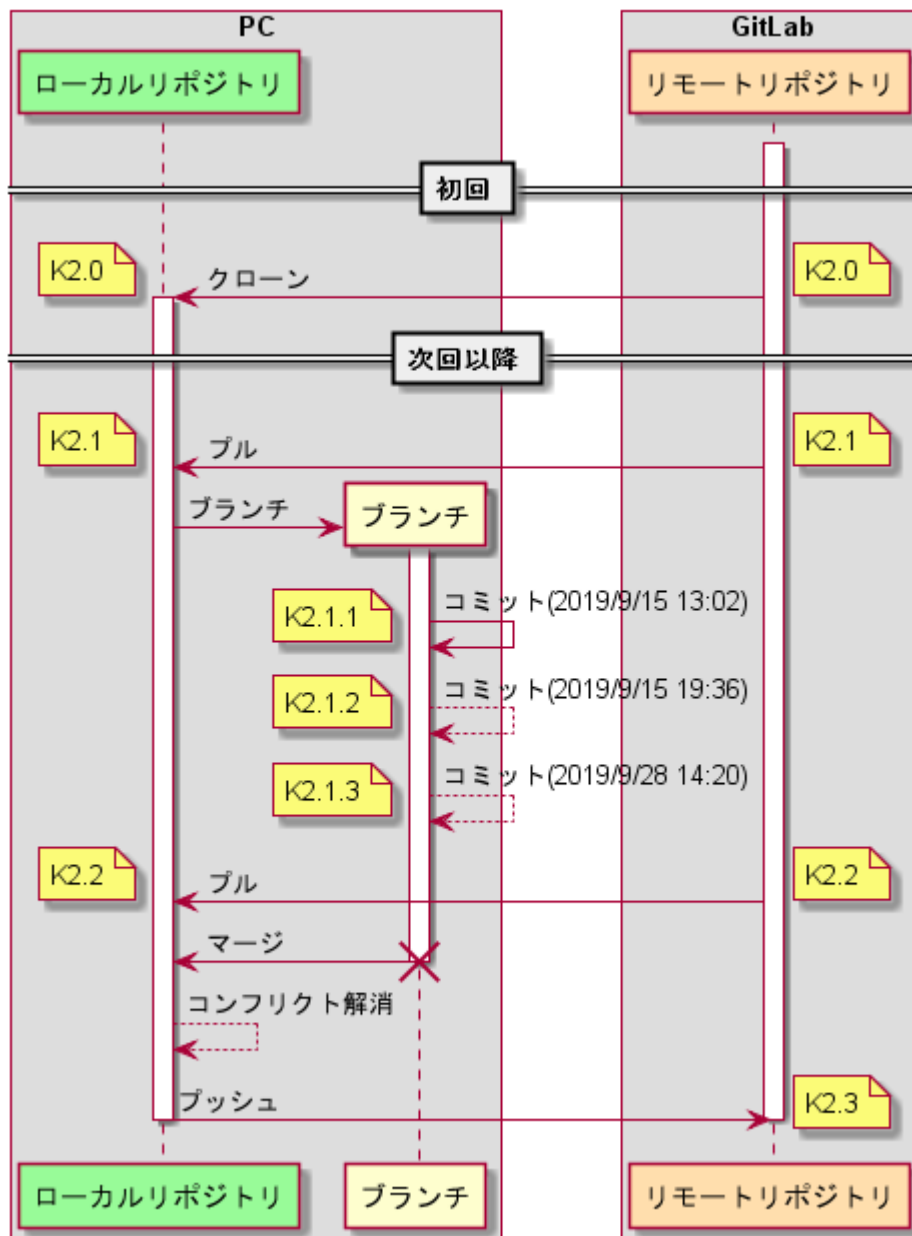
上記のようなやりとりを経て、最終的にマージされるソースコードの品質を高めることが可能です。

## 5.2.4. 運用フロー

初めにリモートリポジトリをローカルに複製(クローン)します。

以降はローカルリポジトリにて変更作業を行っていきますが、自社では以下の手順を遵守することとします。

1. 変更作業を始める前に、最新の状態をリモートリポジトリから**プルし直す**
2. 変更作業はmasterブランチでは行わず、**ブランチを切って移動**して行う
3. 移動先のブランチ内にて、変更作業を行い、**変更箇所ごとに都度コミット**を実行する
4. 全ての変更作業を終えたら、masterブランチに移動し、最新の状態をリモートリポジトリから**プルし直す**
5. 変更作業中のブランチに再び移動し、**masterブランチへのマージ**を行う
6. マージの際に、他の開発者による変更との競合が発生した場合は、**競合内容を確認し修正をコミット**する
7. リモートリポジトリへ**プッシュ**する(または、プルリクエストを行う)



この運用の一番の目的は、**リモートリポジトリは常にリリースできる状態に保つこと**、です。

そのため、**ローカルリポジトリのmasterブランチは最新に保つこと**を心掛け、**編集はブランチを切って行います**。

リモートリポジトリの最新の状態(origin/masterブランチ)より**先行した状態にしてから**リモートリポジトリにプッシュします。

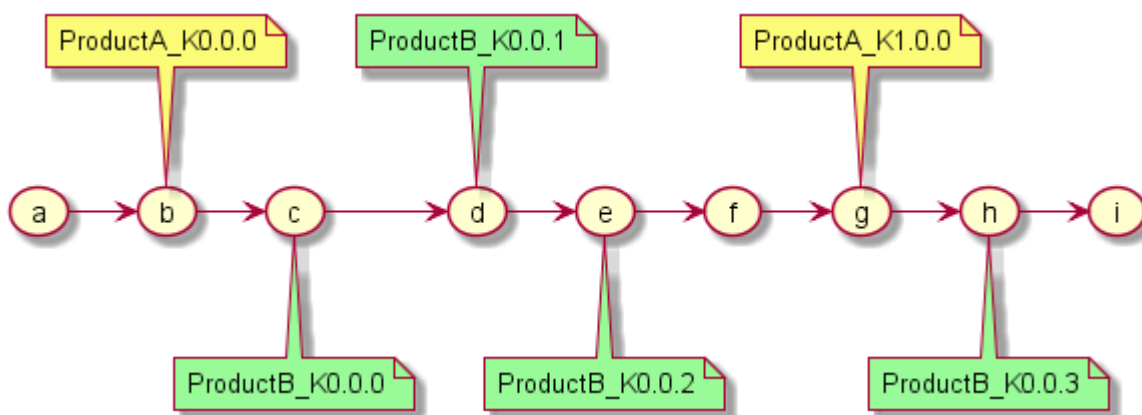
### 5.2.5. ドキュメントのバージョン管理

gitではコミットに対してコミットメッセージとは別にタグを付けることができます(後付け可能)。変更履歴上の重要なポイントへの印として一覧表示で参照も可能で、主にリリースポイントとして使われます。

自社では、タグにバージョンを明記して管理することとします。

タグ記載ルール「a\_Kx.y.z」

- a: プロダクト名
- x: 正式発行ver(0:Draft、1:初期提示、2～:フェーズ移行で発行依頼があれば改版)
- y: 仕様変更ver(0:xが改版時、1～:仕変時に改版)
- z: 開発管理ver(0:yが改版時、1～:仕変前の編集時等に改版)



上記のように、コミットに対してタグを付けて、どのコミットまでの修正がそのプロダクトに適用される仕様かを識別します。

前提として、標準仕様とバイプロ仕様のように仕様書を別管理としていた従来のやり方はやめます。仕様書は一つのドキュメントで管理し、`ifdef` ~ `endif` で条件によって表示/非表示を切り替えることで、プロダクト別の仕様書に早変わりできるような作りにします(`ifdef`の使い方はテストサンプル参照)。

従って、仕様の記載を追加することはあっても削除することはできない(許されない)ので、常に他のプロダクトの仕様を参照しながら作成ができ、仕様の水平展開や標準化を意識した作りにできます。

実際にソフト屋さんに展開する際は、一応体裁を整える意味で、表紙やヘッダーに付く改訂情報も合わせて修正して、PDF化して展開することとします。

例)

```

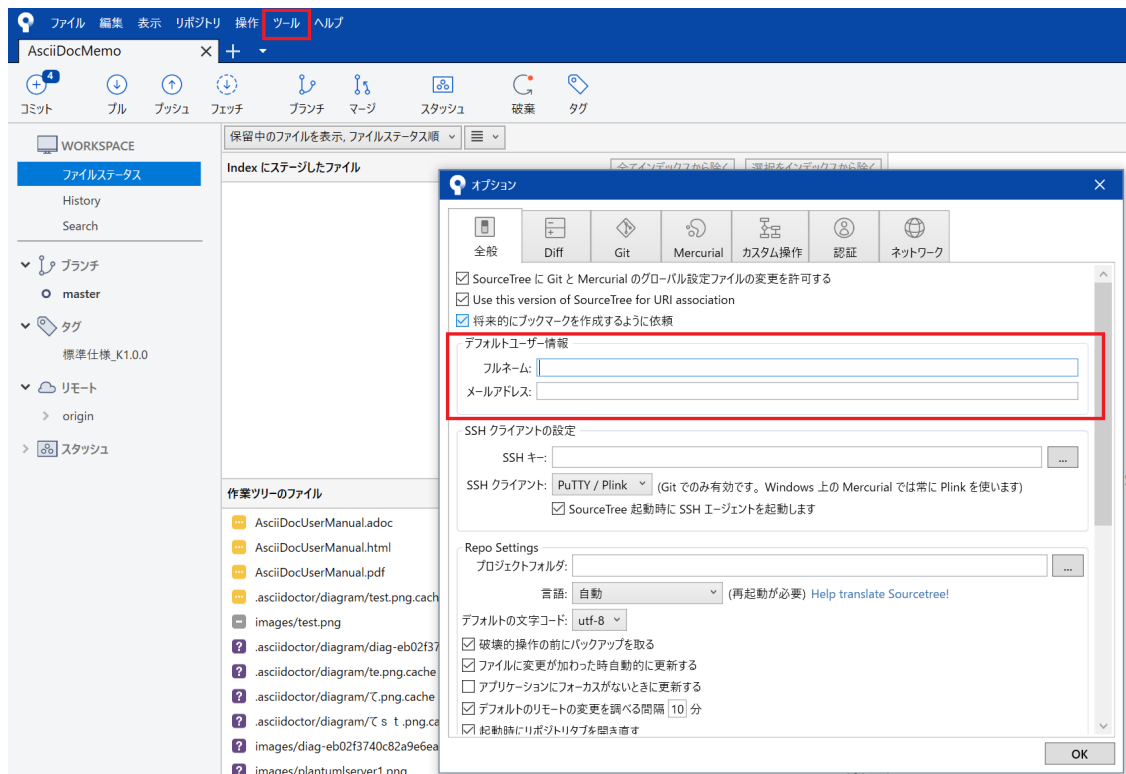
//ドキュメントタイトル、表紙に入る
= AsciiDoc ユーザーマニュアル
//ドキュメントタイトル、ヘッダーに入る
:docname: AsciiDoc ユーザーマニュアル
//改定番号
:revnumber: K1.0.0
//改定日
:revdate: 2019/10/15

```

## 5.2.6. SourceTreeを使って運用する

1. SourceTreeにGitLabアカウントを登録します。

[ツール]→[オプション]のデフォルトユーザー情報に、GitLabで登録したユーザー名とメールアドレスを入力します。



2. これまで説明してきた作業は、GitのGUIツールであるSourceTreeを使用していますが、以下の参考サイトに初心者でもわかるようにまとまっているので説明は割愛します。

<誰でも簡単!GitHubで管理するためのSourcetreeの最低限の使い方>

<https://haniwaman.com/sourcetree/>

<SourceTreeの使い方|初心者が習得すべき基本操作>

<https://ics.media/entry/1365/>



(お役立ち情報) 外部diffを使って、コミット間の差分を見やすく表示する方法

<https://blog.stedplay.com/how-to-use-sourcetree-with-p4merge/>

※本手順書内ではパッチファイルで、P4Mergeの代わりにWinmergeをインストール済みです

以上で終わりです!