

AsciiDoc ユーザーマニュアル

K1.0, 2019/10/25

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	12
4.5. VScodeの拡張機能をインストール	13
5. 使い方	15
5.1. VScodeで実際にAsciiDocを書いてみる	15
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)
- 表現力も高く技術書作成に向いている(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
 - 署線、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)
 - 記法が少ないの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
 - 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も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(可読性は保つつつ、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

- 以下のバッチファイルをダブルクリックで実行

```
①ChocolateyInstall.bat
```

- 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
- 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

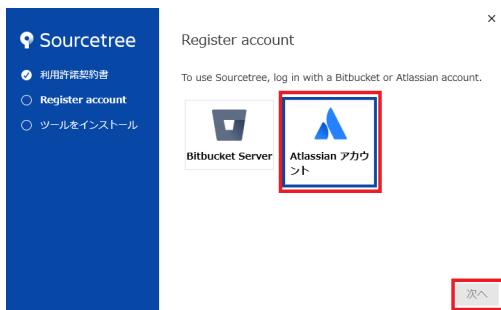
- 以下のバッチファイルをダブルクリックで実行

```
②ChocolateyPackageInstall.bat
```

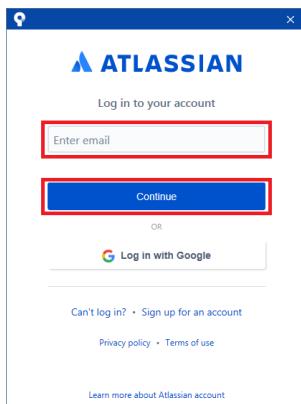
- ユーザーアカウント制御の許可のポップアップが出るので **はい** をクリック
- コマンドプロンプトが表示されて処理が進むのでしばらく待つ(20分程度かかります)
- 下記画面が表示されたら **ライセンスに同意します** にチェックを入れて **次へ** をクリック



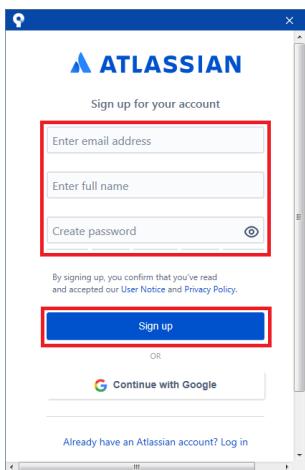
- Atlassianアカウント** を選択して **次へ** をクリック



- メールアドレスを入力して **Continue** をクリック



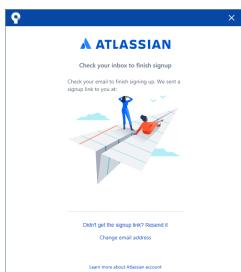
- 任意のユーザー名とパスワードを入力して、**Sign up** をクリックで、アカウントを作成



8. 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



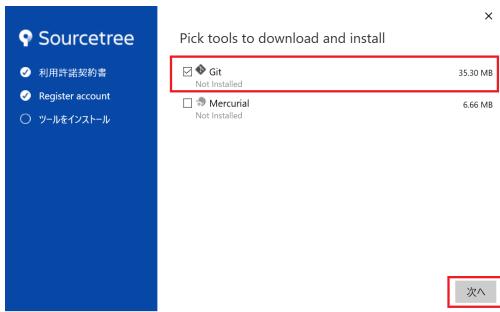
9. メールアドレスの認証メールが届くので、メール内の認証ボタンをクリック後、下の画面を閉じる



10. 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



11. 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



12. Gitのダウンロードが完了したら、**次へ** をクリック



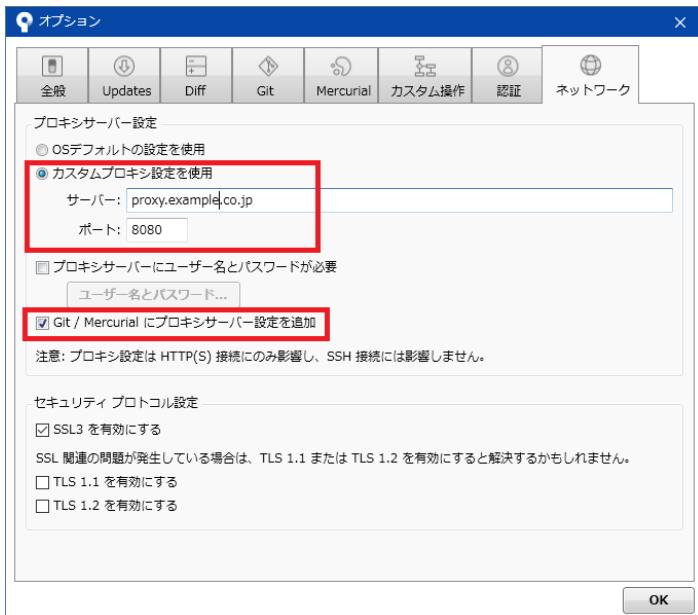
- 社内のProxy環境化の場合、ダウンロードに失敗してエラーとなるので、一度インストール画面を閉じる
- Windowsのスタートメニューから **Sourcetree** を検索して起動
- 下記画面が表示されたら、一番下の選択肢の **Gitを使いたくない** をクリック



- Sourcetreeが起動するので、[ツール]→[オプション]を開く



- 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



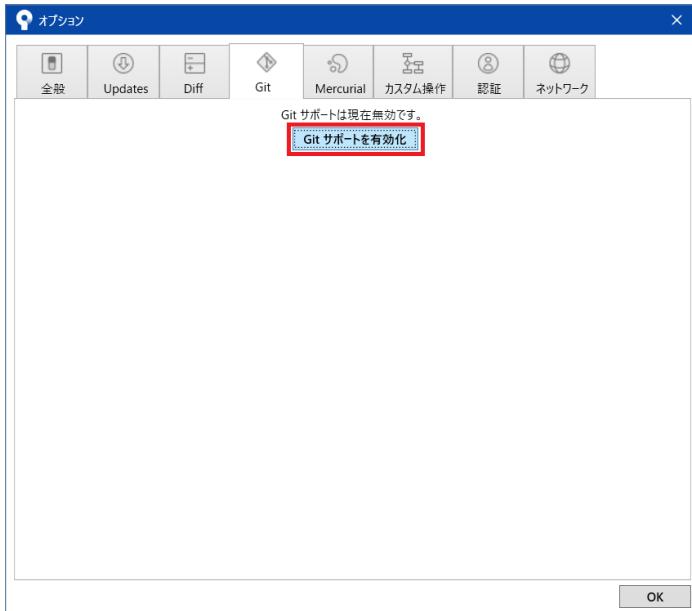
proxyの確認方法(Windows10の場合)

<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/>

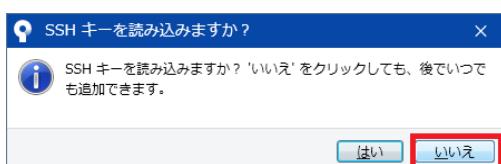
f. 続けて、Git タブを選択し、**Gitサポートを有効化** をクリック



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

13. 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



14. Sourcetreeが起動したら、一旦閉じる

15. 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10まで完了していれば **y**、完了していない場合は **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

- 以下のバッチファイルをダブルクリックで実行

```
③AsciiDocToolInstall.bat
```

- 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

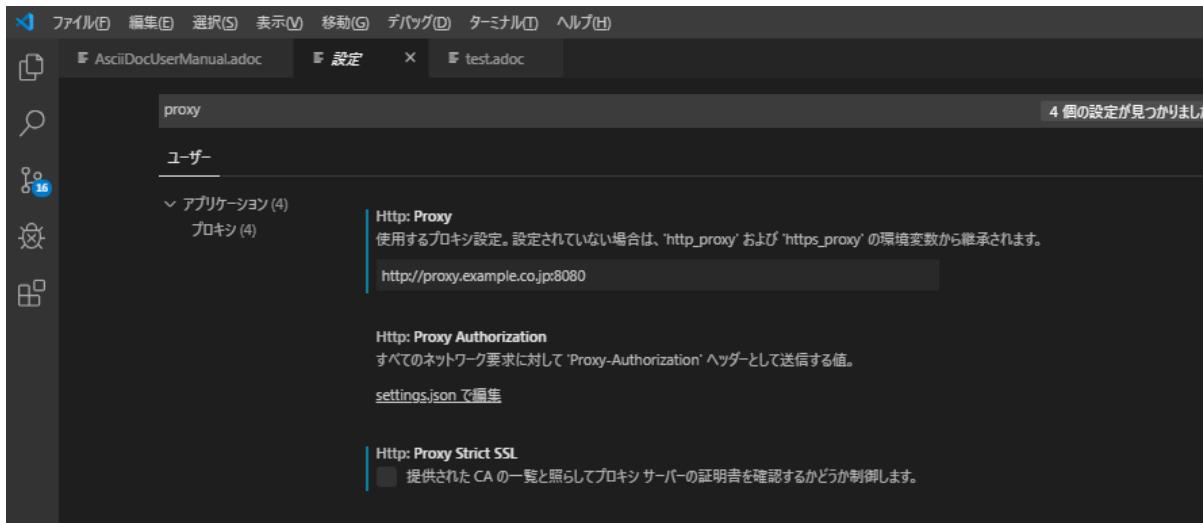
```
gem install xxxx -p http://アドレス:ポート
```

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+P** 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>

VScodelにProxyを設定する方法
<https://qiita.com/coinloss1973/items/b3c84daeed90fd183501>

※最後にProxyの設定を元に戻すのは、この先の手順でPlantUMLサーバーで同じポートを使うため

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ペして利用

```
// 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ペして利用

```
// 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、以下のいずれかの対応でもOK

- :**stylesdir**: をアトリビュートに指定
- VScodeの設定で、**Asciidoc > Preview: UseEditorStyle** のチェックを外す

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作成し使用

Styleについて

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



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

```
// ruby2.6でasciidocor-pdfのverが1.5.0.beta.2の場合  
C:\tools\ruby26\lib\ruby\gems\2.6.0\gems\asciidocor-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/  
        └ asciidocor-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

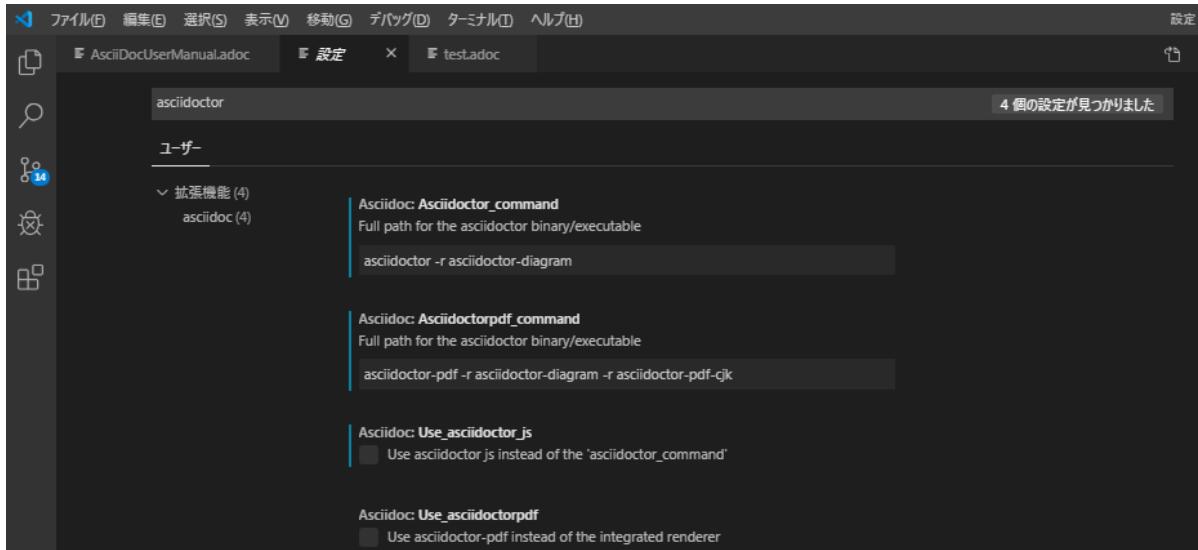
[ファイル]→[ファイルを開く]から **template** フォルダ内の **test.adoc** ファイルを開きます。

テストサンプルをプレビューする

asciidocの設定を変更する(初回のみ実施)

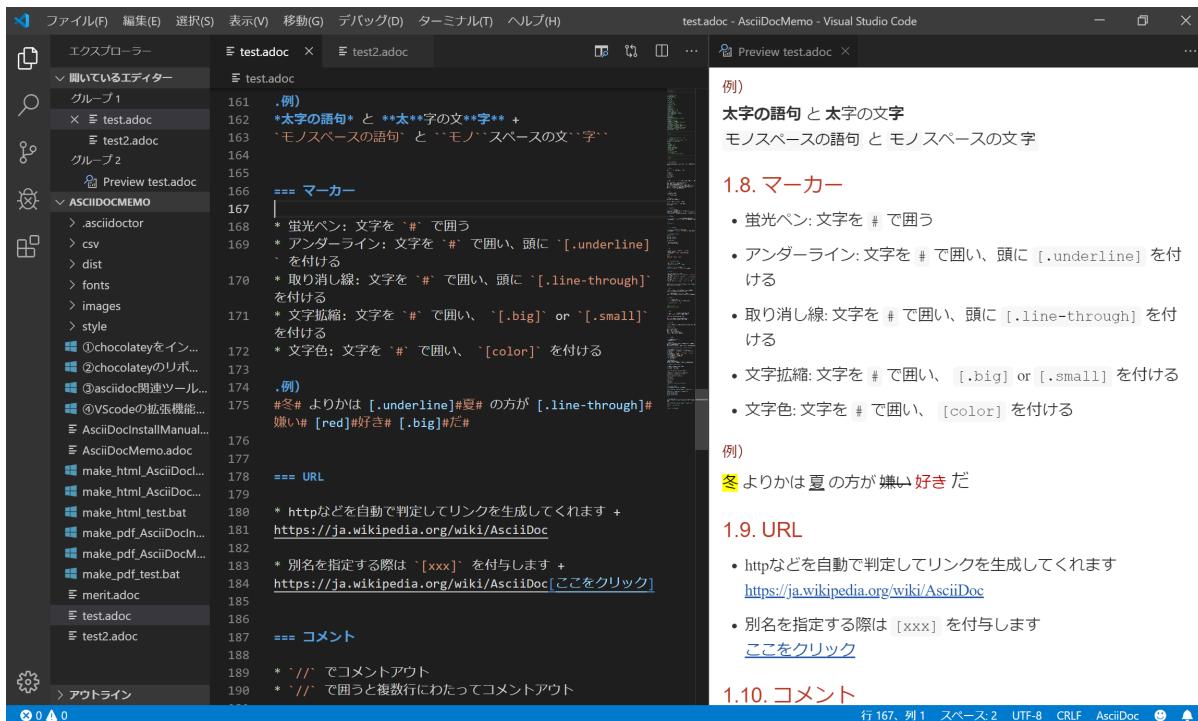
- ショートカット **Ctrl+Shift+S** で設定を開き、**asciidoc** を検索し、以下を設定

```
"AsciiDoc.asciidoctor_command": "asciidoc -r asciidoctor-diagram",
"AsciiDoc.asciidoctor_command": "asciidoc-pdf -r asciidoctor-diagram -r
asciidoc-pdf-cjk",
"AsciiDoc.use_asciidoctor_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への対策)

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. Proxy環境化の場合、MavenにProxyの設定を行う
 - a. C:\ProgramData\chocolatey\lib\maven\apache-maven-3.6.2\conf にある `Settings.xml` を適当なテキストエディタ(VScodeで良い)で開く
 - b. 以下の部分を編集してProxyの設定を行う(Proxyの確認手順は先述の通り)
 - i. 変更前

```
<proxies>
    <!-- proxy
    | Specification for one proxy, to be used . . .
    |
    <proxy>
        <id>optional</id>
        <active>true</active>
        <protocol>http</protocol>
        <username>proxyuser</username>
        <password>proxypass</password>
        <host>proxy.host.net</host>
        <port>80</port>
        <nonProxyHosts>local.net| . . . </nonProxyHosts>
    </proxy>
    -->
</proxies>
```

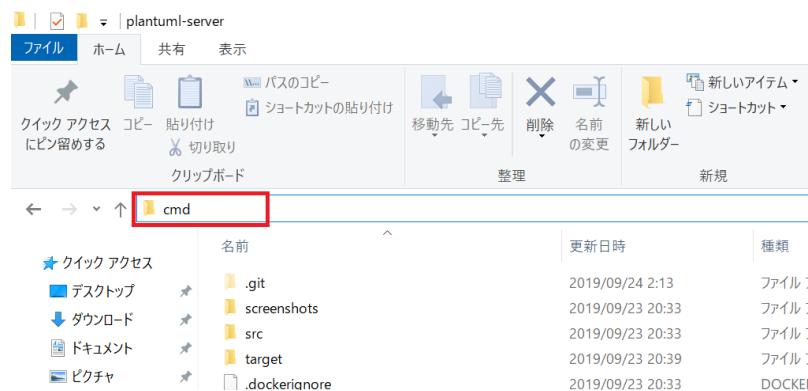
- ii. 変更後

```
<proxies>
    <proxy>
        <id>optional</id>
        <active>true</active>
        <protocol>http</protocol>
        <host>proxy.example.co.jp</host>
        <port>8080</port>
    </proxy>
</proxies>
```

2. 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>

3. コマンドでサーバーを立ち上げる

- 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
- 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.
```

エラー:Address already in use が発生した場合

- コマンドプロンプトで8080番を使っているプロセスがないか確認する

```
netstat -aon | find "8080"
TCP x.x.x.x:8080 LISTENING aaaa
```

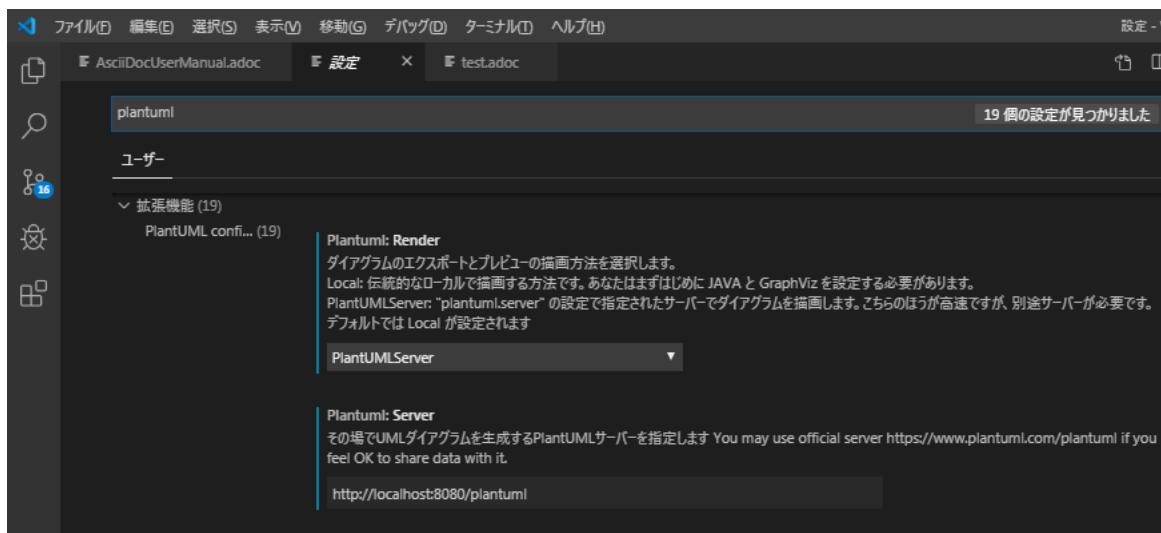


- 上記の場合、aaaaのプロセスが8080番ポートを使ったままということなのでプロセスを終了させる

```
taskkill /F /pid aaaa
成功: PID aaaa のプロセスは強制終了されました。
```

4. VScodeにて、ショートカット `Ctrl+Shift` 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",
```



5. UML図の編集

- 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
--
```

- 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

- 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
```

- 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)
- 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  
asciidoc -r asciidoc-diagram -o *.html *.adoc ②  
asciidoc-pdf -r asciidoc-diagram -r asciidoc-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

下記のサイトを読むと良い。

- Gitとは？

[サル先生のGit入門～バージョン管理を使いこなそう～](#)

- ただ、自分はサル以下かも知れないとと思うので、まずはこのエントリーを読むと良い。

[「サルでもわかる」が分からなかった時に読むGit、SourceTreeの超訳](#)

以下に上記サイトから抜粋した各章と、それに対する補足、対応するSourceTreeでの操作について説明する。
一行ずつ参照リンクが異なるので、順番にクリックして読むと良い。

<入門編>

Gitの基本：

[Gitを使ったバージョン管理](#)

- バージョン管理がしっかりとできる、というだけでなく、オリジナルのファイルの格納場所が一意に定まる（後述するリモートリポジトリになる）という点も、仕様書の所在など管理が全くできていないチームには有効だと思われる（自虐）。

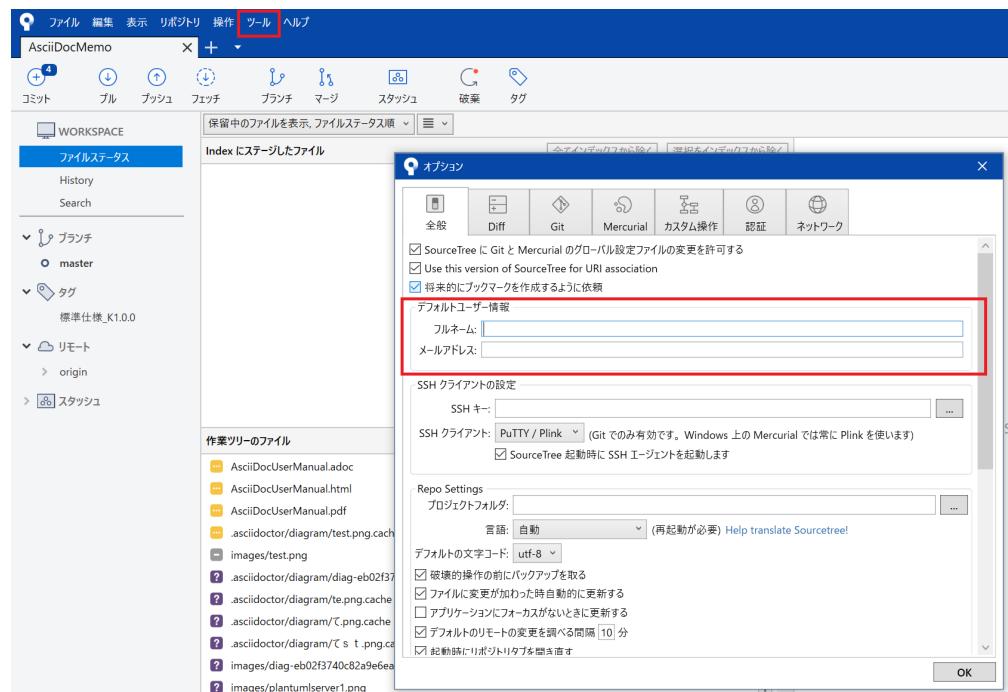
[履歴を管理するリポジトリ](#)

- リモートリポジトリとして、Gitを利用した開発者を支援するホスティングサービスであるGitLabを利用する。
- リモートリポジトリには、URL (<https://gitlab.xxx/yyy>) でアクセス可能。
- リモートリポジトリをサーバ上に置き、開発者それぞれが作業のためのリポジトリをローカルに持つという構成は、分散型と呼ばれる。
- 分散型では、共有しているリモートリポジトリを汚すことなく、またリモートリポジトリにアクセスできない環境（ネットに繋がらない等）でもローカルで作業を進めることができる。

[変更を記録するコミット](#)

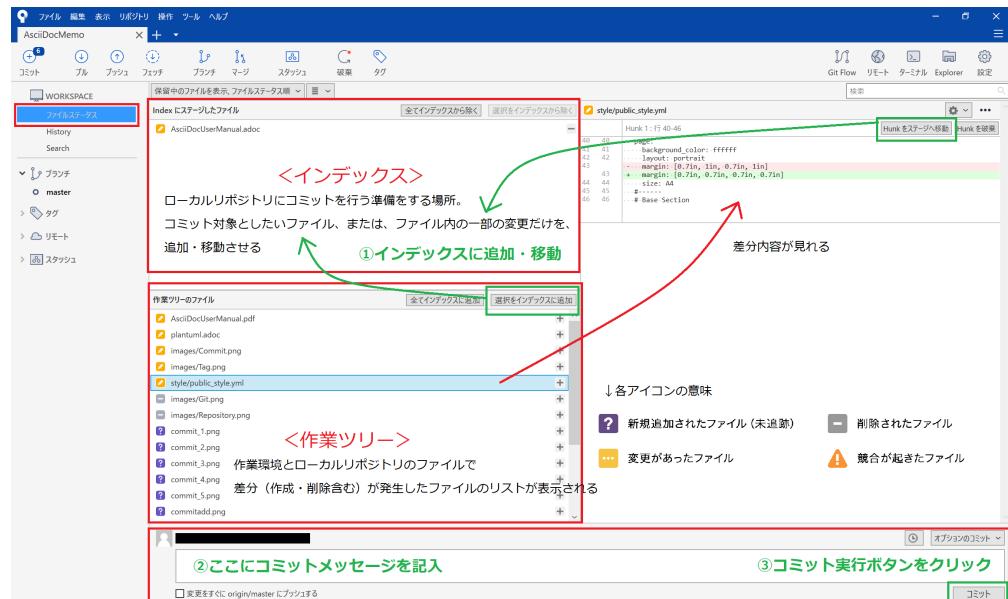
- 誰がコミットしたのかわかるようにSourceTreeにGitLabアカウントを登録

[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



作業ツリーとインデックス

- SourceTree上でコミット操作



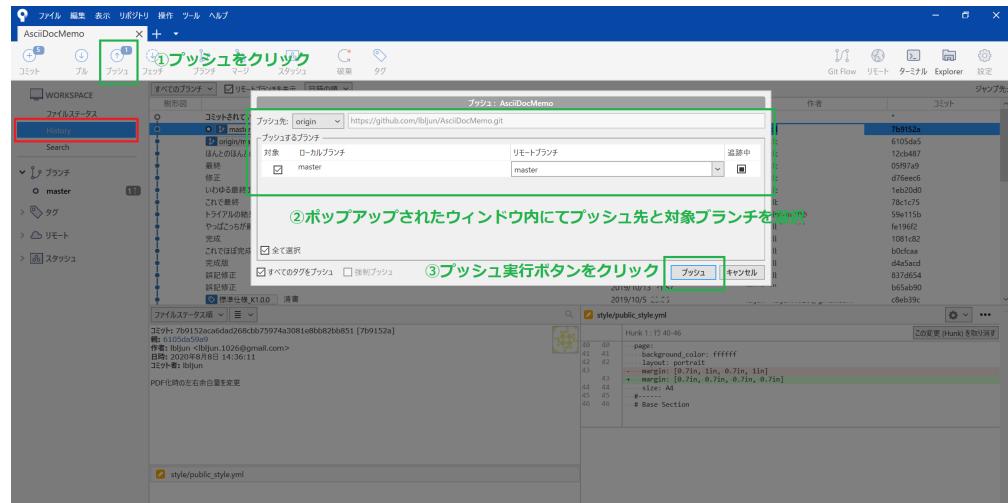
- 任意のファイルだけを追加(選択をインデックスに追加)
- 任意のファイル内的一部の変更だけを追加(Hunkをステージへ移動)
- コミットメッセージは、"なぜ(Why)" 変更したかがわかるように付ける

リポジトリの共有:

リモートリポジトリにプッシュする

- SourceTree上でプッシュ操作

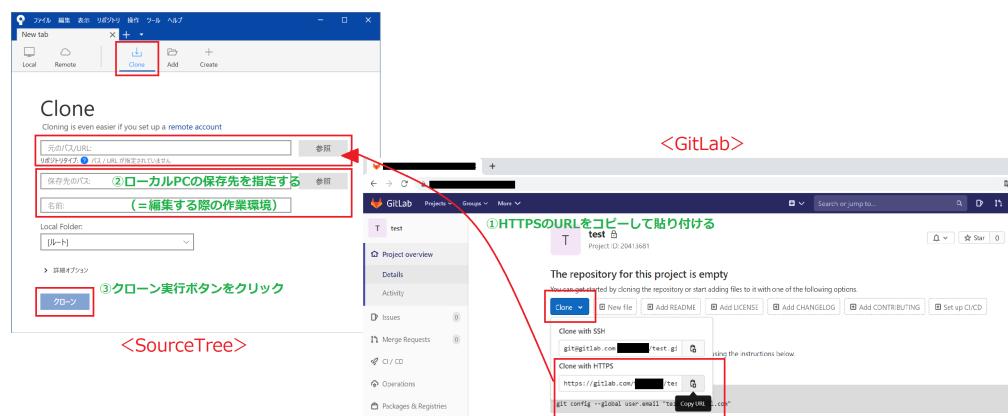
(後述の、ブランチを作成して運用する、ということをやらない限りは、②の選択はデフォルトのままでOK)



リモートリポジトリをクローンする

- SourceTree上でクローン操作

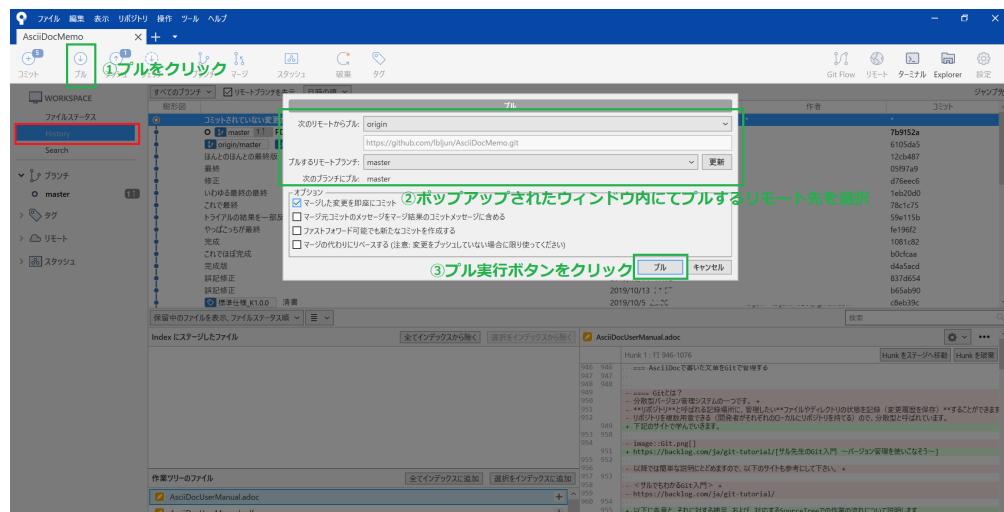
(GitLabからクローン、実施必要なのはプロジェクト毎(=リポジトリ毎=仕様書パッケージ毎)に初回のみ)



リモートリポジトリからプルする

- SourceTree上でプル操作

(後述の、ブランチを作成して運用する、ということをやらない限りは、②の選択はデフォルトのままでOK)



- ただし、ブランチを作成して運用する場合は、このプル操作は使わない方が良い。

理由は、ローカルリポジトリとリモートリポジトリ間の操作において、**プッシュの反対はプル"ではない"**から（詳細はブランチの章にて説明）。

変更履歴の統合：

※まあ、一人さみしく仕様書作成している限りは以下は発生しない（自虐）。

変更履歴のマージ

- [SourceTree上でのマージ操作](#)

競合の解決

- [SourceTree上での競合の解決操作](#)

<発展編>

※複数人で並行して同一ファイルを修正するようなケースが有り得る場合や、きちんと修正内容のレビュー/検査をして責任者や検査者にリモートリポジトリoriginのmasterブランチに反映してもらうような運用をする場合には、ローカルでブランチを作成して（切って）作業する。てか、そうすべき。

ブランチ：

ブランチとは

ブランチの運用

- [SourceTree上でのブランチ操作](#)

- 追跡ブランチとリモートブランチは別物！

- 追跡ブランチのもっと正確な言葉の使い方
- 追跡ブランチとフェッチとマージの関係
- プルではなくフェッチとマージを使え

ブランチの切り替え

- [SourceTree上でのブランチの切り替え（チェックアウト）操作](#)

- リモートにあるブランチが表示されてなければフェッチ

- ローカルに残ってしまっているリモートブランチを一括削除

- [GitのHEADとは何者なのか](#)
- [Gitの"detached HEAD"状態とその注意点](#)
- [SourceTree上でのスタッッシュ操作](#)
 - 特定のコミット間の差分を見るのに便利なdiff操作も見ておくと良い。
 - [さらに外部diffを使ってさらに見やすく表示する](#)

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

[ブランチの統合\(マージ、リベース\)](#)

- [SourceTree上でのマージ、リベース操作](#)

[トピックブランチと統合ブランチでの運用例](#)

- ゴリゴリのソフトウェア開発じゃなければMasterとDevelop(ここで作業)があれば十分

リモートリポジトリ:

※ブランチの概念を理解した上で基本編のおさらい

[プル](#)

[フェッチ](#)

[プッシュ](#)

タグ:

[タグ](#)

- [SourceTree上でのタグ付け操作](#)

コミットの書き換え:

[直前のコミットを修正する](#)

[過去のコミットを打ち消す](#)

[コミットを捨てる](#)

[コミットを抜き取る](#)

[コミットの履歴を書き換える](#)

[ブランチ上のコミットを一つにまとめてマージする](#)

- [SourceTree上でのコミット書き換え操作](#)

プルリクエスト:

[プルリクエストとは?](#)

[プルリクエストのメリット](#)

[プルリクエストを使った開発プロセス](#)

- [GitLab上でのマージリクエスト\(プルリクエスト\)操作](#)
- [GitLab上でのレビュー・承認操作](#)

マージできない場合は?:

※ブランチの概念を理解した上で基本編のおさらい

[競合の発生](#)

[競合の解決](#)

以上で終わりです!