

Visual Studio Code を 活用するための 人気 TIPS 12 選

Insider.NET 編集部 かわさきしんじ [著]

01.VS Code で Markdown をプレビューするには？

02.VS Code で Markdown を HTML や PDF に変換するには？

03.VS Code で表示言語を変更するには

04.VS Code を持ち運ぶには（ポータブルモード）

05.VS Code でウィンドウサイズを制御するには

06.VS Code で空白文字を一目で分かるように表示するには

07.VS Code をアンインストールするには

08.VS Code のミニマップの表示／非表示を切り替えるには

09.VS Code のミニマップの表示方法を変更するには

10.VS Code でフォントを変更するには

11.VS Code のコマンドラインオプション：起動編

12.VS Code のコマンドラインオプション：拡張機能編

初出：@ IT Insider.NET フォーラム

【注意】

- eBook 化に際して、上記の記事を一部加筆修正しています。
- 本書に記載されている社名・商品名は一般に各社の商標または登録商標です。
- その他、免責事項については@ IT Web サイトのポリシーに準拠します。

<http://www.atmarkit.co.jp/aboutus/copyright/copyright.html>

VS Code で Markdown をプレビューするには？

Visual Studio Code（以下、VS Code）はデフォルトで Markdown 記法を使って記述されたドキュメントをサポートしている。Markdown ドキュメントをプレビューする方法を説明する。

操作	Windows/Linux	macOS
サイドバイサイドにプレビューを表示	[Ctrl] + [K] → [V]	[Command] + [K] → [V]
同一エディタに別タブとしてプレビューを表示	[Ctrl] + [Shift] + [V]	[Shift] + [Command] + [V]
プレビューを特定ファイルにロック	コマンドパレットで [Markdown: Toggle Preview Locking] を選択	

Markdown のプレビューに関するキー操作やコマンド

Markdown をサイドバイサイドにプレビューするには？

VS Code が標準で提供する機能を使えば、現在編集中の Markdown をプレビューできる。一般的には次の方法を使うとよいだろう。

プレビューをサイドバイサイド（隣り合わせ）に表示

- コマンドパレット：[Markdown: プレビューを横に表示] (Markdown: Open Preview to the side)
- キーボード：[Ctrl] + [K] → [V] (Win / Linux)、[Command] + [K] → [V] (macOS)

「プレビューをサイドバイサイド（隣り合わせ）に表示」するには、プレビューしたい Markdown を開いているエディタ（のタブ）をアクティブにして、上に示したコマンドパレットのコマンドやキー操作を行えばよい。すると、Markdown を編集しているエディタ *1 とは別のエディタが隣に開き、そこにプレビューが表示される。

The screenshot shows the Visual Studio Code interface with two panes. The left pane displays the Markdown file 'sample01.md' containing code snippets and a preview of the rendered content. The right pane shows the rendered preview of the Markdown file. The status bar at the bottom indicates the file has 9 lines and 1 character, uses UTF-8 encoding, and is in CRLF mode.

```

1 # Visual Studio Code TIPS
2 ## VS CodeでMarkdownをプレビューするには？
3
4
5
6 Insider.NET編集部 かわさき しんじ
7 2018/04/20
8
9
10 Visual Studio Code (以下、VS Code) はデフォルトでMarkdown記法を使って記述されたドキュメントをサポートしている。Markdownドキュメントをプレビューする方法を説明する。
11
12 |操作|Windows/Linux/macOS|
13 |---|-----|
14 |サイドバイサイドにプレビューを表示| [Ctrl] + [K] → [V] | [Cmd] + [K] → [V] |
15 |同一エディタに別タブとしてプレビューを表示| [Ctrl] + [Shift] + [V] | [Shift] + [Cmd] + [V] |
16 |プレビューを特定ファイルにロック| コマンドパレットで [Markdown: Toggle Preview Locking] を選択|←|
17 **Markdownのプレビューに関するキー操作やコマンド**
18
19
20 ### Markdownをサイドバイサイドにプレビューする

```

プレビューを隣り合わせで表示したところ

編集している Markdown とそのプレビューは基本的には同期して表示される。内容を変更すれば、それがリアルタイムにプレビューにも反映されるし、一方をスクロールすれば、それに合わせてもう一方もスクロールされる。

*1 VS Code では実際にファイルを開いて編集する部分を「エディタ」と呼び、そこには複数のファイルをタブ形式で表示できる。エディタは最大で 3 つまで並べて表示できる。個々のエディタ（とそこに開かれたファイル群）を「エディタグループ」と呼ぶこともある。

なお、コマンドパレットやキーボードを使わずに、GUI ベースで操作をするなら、Markdown ファイルを開いているエディタ（タブ）の右側にある [プレビューを横に表示] ボタンをクリックしてもよい。



[プレビューを横に表示] ボタン

プレビューを同じエディタの別のタブに表示するには？

[Ctrl] + [Shift] + [V] キー (macOS では [Command] + [Shift] + [V] キー) を押せば、同じエディタ内に別のタブとしてプレビューを表示することもできる。

The screenshot shows the Visual Studio Code interface with the title bar "プレビュー sample01.md - vsctips0001 - Visual Studio Code". The menu bar includes "ファイル(F)", "編集(E)", "選択(C)", "表示(V)", "移動(G)", "デバッグ(D)", "タスク(T)", and "ヘルプ(H)". The left sidebar has icons for file, search, and other functions. The main editor area displays the content "Visual Studio Code TIPS" and "VS CodeでMarkdownをプレビューするには？". Below the editor, it says "Insider.NET編集部 かわさき しんじ 2018/04/20". A horizontal line separates this from the preview section. The preview section starts with "Visual Studio Code (以下、VS Code) はデフォルトでMarkdown記法を使って記述されたドキュメントをサポートしている。Markdownドキュメントをプレビューする方法を説明する。". It contains a table comparing keyboard shortcuts for different preview modes across Windows/Linux and macOS.

操作	Windows/Linux	macOS
サイドバイサイドにプレビューを表示	[Ctrl] + [K] → [V]	[Cmd] + [K] → [V]
同一エディタに別タブとしてプレビューを表示	[Ctrl] + [Shift] + [V]	[Shift] + [Cmd] + [V]
プレビューを特定ファイルにロック	コマンドパレットで [Markdown: Toggle Preview Locking] を選択	-

Markdownのプレビューに関するキー操作やコマンド

Markdownをサイドバイサイドにプレビューするには？

VS Codeが標準で提供する機能を使えば、現在編集中のMarkdownをプレビューできる。一般的には次の方法を使うのよいだろう。

同一エディタ内に別のタブとしてプレビューを表示したところ

先ほどとは異なり、エディタが同じなので、隣り合わせで表示されない。ソースとなる Markdown に移るには、コマンドパレットから [Markdown: ソースの表示] ([Markdown: Show Source]) コマンドを選択する。あるいは、以下のように、プレビューしているエディタ（タブ）の右端にある [ソースの表示] ボタンを利用してよい。

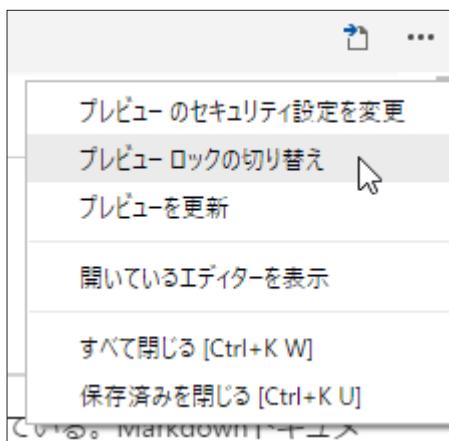


[ソースの表示] ボタン

プレビューを特定のファイルにロックするには？

複数の Markdown を編集している場合、デフォルトでは、プレビュー用のエディタ（タブ）には、現在編集しているファイルの内容が表示される（別のファイルに切り替えれば、そのプレビューが表示される）。これを特定のファイルにロックすることもできる。これには、コマンドパレットで [Markdown: プレビュー ロックの切り替え] ([Markdown: Toggle Preview Locking]) コマンドを実行する。これにより、プレビューには特定のファイルの内容が常に表示されるようになる。

以下のように、プレビューをしているエディタ（タブ）の右側にある [...] ボタンをクリックして、ポップアップメニューから [プレビュー ロックの切り替え] を選択してもよい。



[...] ボタンをクリックすると表示されるポップアップメニュー

なお、このポップアップメニューにある Markdown のプレビュー関連の操作としては以下のものがある。

- [プレビュー のセキュリティ設定を変更] : VS Code では安全でないコンテンツ（HTTPS を使用していない外部サイトにある画像ファイルへのリンク、スクリプトなど）を表示しないようになっている。この設定を変更するのに使用する
- [プレビューを更新] : 画像ファイルを入れ替えるなどしたので、明示的にプレビューを更新したい場合に利用する

コマンドパレットには、これらのボタンに対応するコマンドも用意されているので、興味のある方は調べてみよう。

VS CodeでMarkdownをHTMLやPDFに変換するには？

Markdown PDF 拡張機能を使用して、Markdown テキストを HTML や PDF に変換する方法や、ファイル保存時に自動変換を行うための設定を説明する。

Visual Studio Code（以下、VS Code）はデフォルトで Markdown 記法を使って記述されたドキュメントをサポートしている。が、それを HTML や PDF に変換する機能までは持っていない。Markdown を HTML や PDF に変換するには、拡張機能を利用するのが簡単だ。

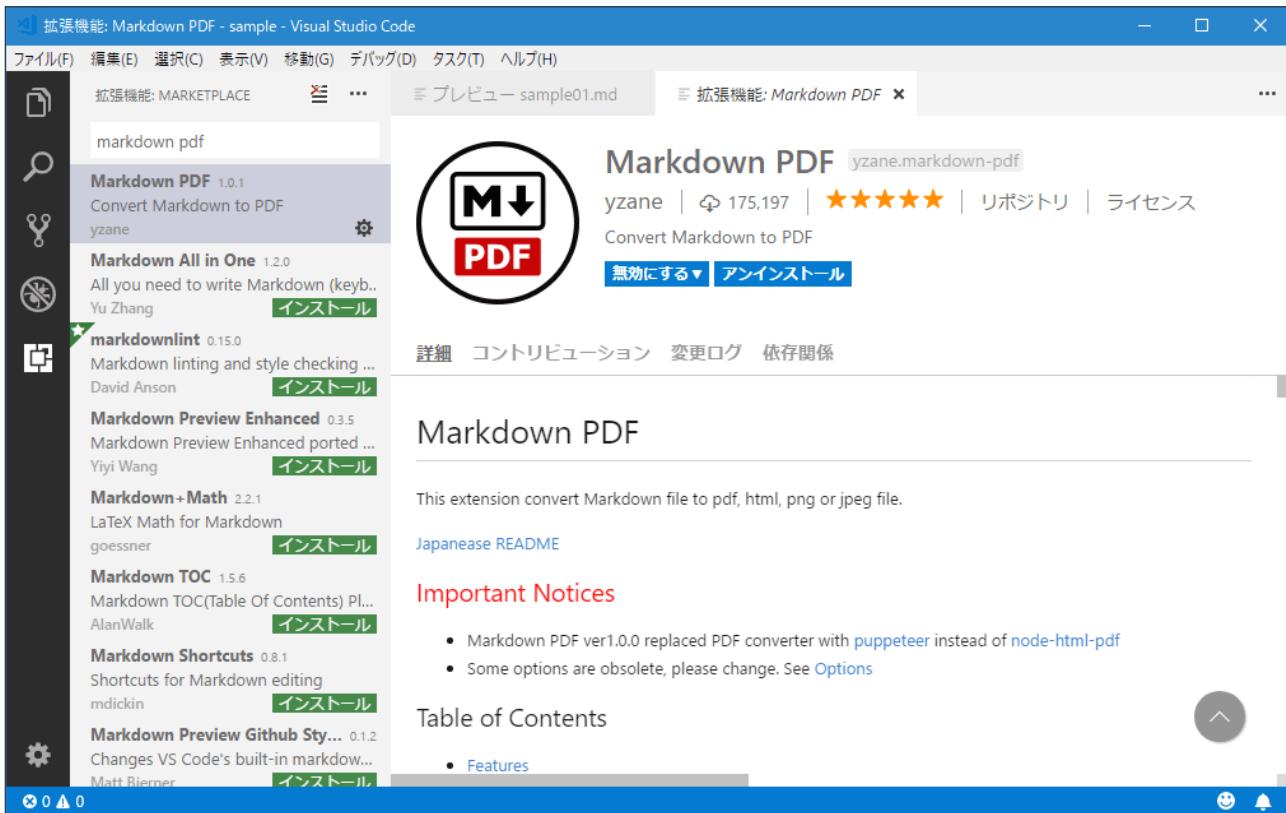
処理	操作
MarkdownをHTMLやPDFに変換する	「Markdown PDF」拡張機能をインストールして、コマンドパレットから [Markdown PDF: Export (html)] コマンドや [Markdown PDF: Export (pdf)] コマンドを実行
ファイルの保存と同時にHTMLやPDFへ変換する	markdown-pdf.type項目で出力フォーマットを指定して、markdown-pdf.convertOnSave項目をtrueにする（VS Codeの再起動が必要）

Markdown を PDF や HTML に変換するための操作

Markdown PDF 拡張機能

VS Code には Markdown を HTML や PDF に変換する機能は備わっていない。そのため、例えば、HTML に変換するには [markdown-it](#) などのツールを手作業でインストールして、「Compiling Markdown into HTML」ページに書かれている手順を自分で実行してもよいが、拡張機能を使った方が簡単だ。

Markdown を HTML や PDF に変換してくれる拡張機能としては [yzane](#) 氏による「[Markdown PDF](#)」がある。この拡張機能をインストールしてしまえば、後はコマンドパレットから [Markdown PDF: Export (XXX)] コマンドを実行するだけで、HTML や PDF、PNG、JPEG 形式のファイルに変換できる（「XXX」には変換先のフォーマットに応じて「html」「pdf」などが入る）。

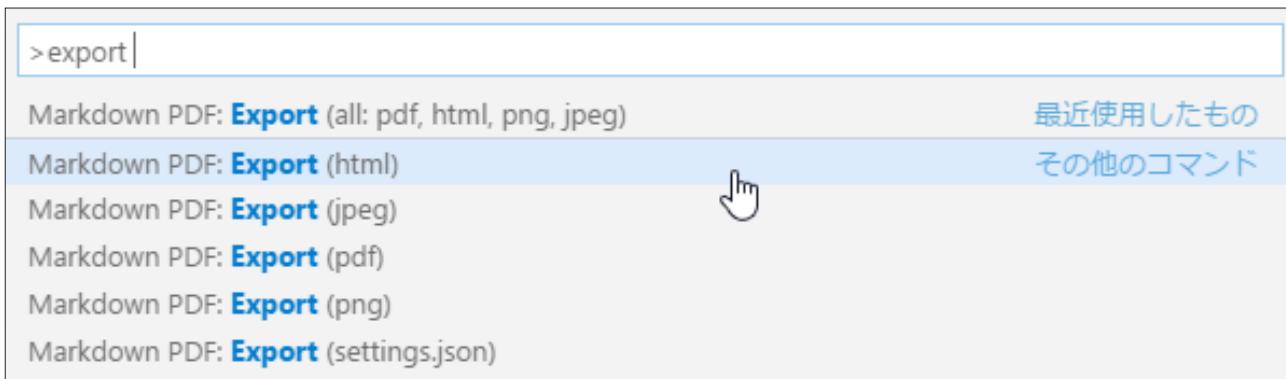


Markdown PDF 拡張機能

まずはこの拡張機能をインストールしておこう。

Markdown を HTML や PDF に変換するには？

HTML に変換したい Markdown を VS Code 上でアクティブにしてから、コマンドパレットで「export」などと入力し、[Markdown PDF: Export (html)] コマンドを実行すればよい。

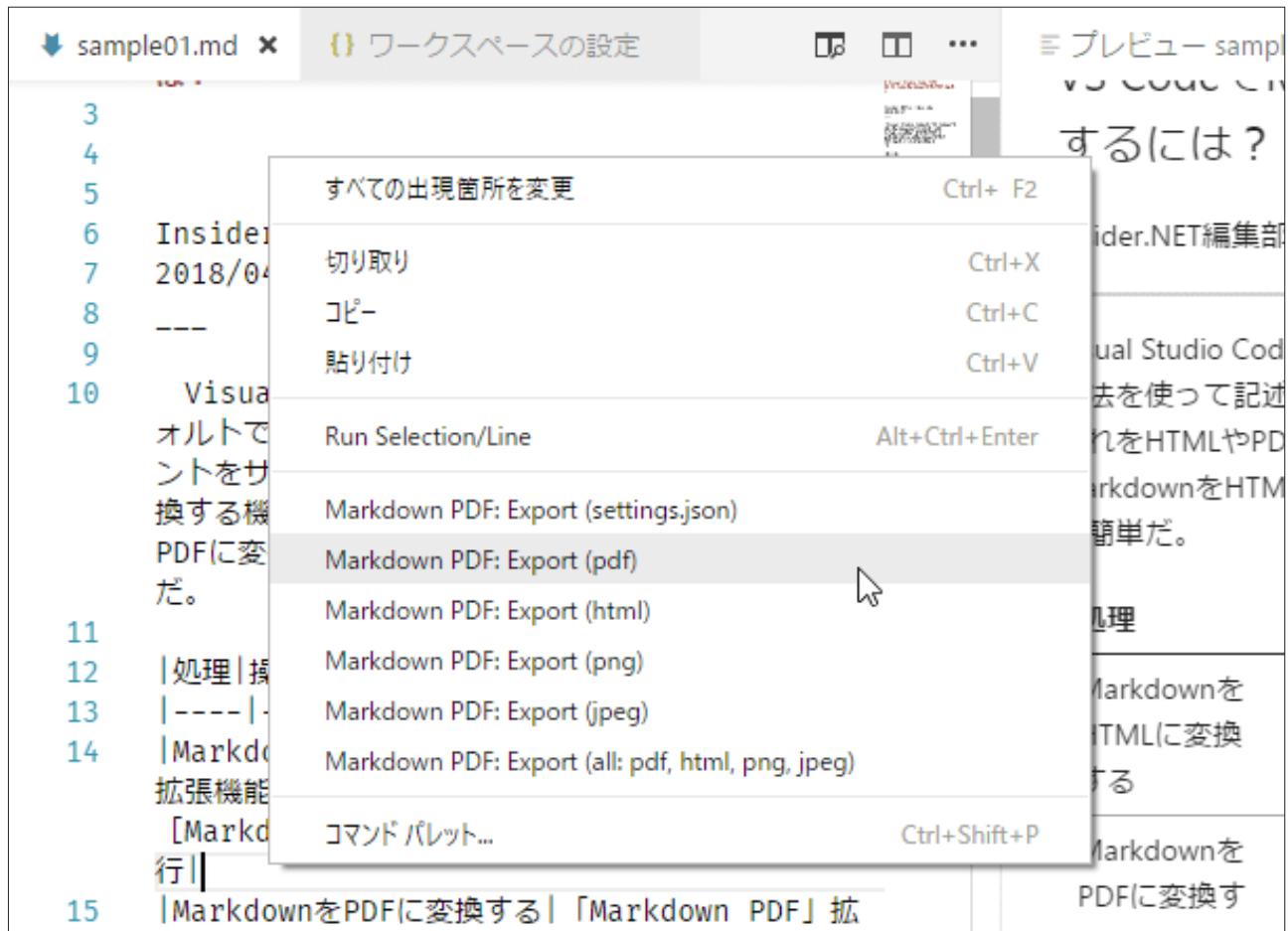


コマンドパレットから [Markdown PDF: Export (html)] コマンドを実行

PDF に変換するには、もちろん [Markdown PDF: Export (pdf)] コマンドを実行すればよい。

なお、上の画像を見ると分かる通り、Markdown PDF は HTML、PDF 以外にも PNG / JPEG 形式のファイルへの変換もサポートしている。[Markdown PDF: Export (all: pdf, html, png, jpeg)] コマンドは全ての出力形式への変換を行うもので、[Markdown PDF: Export (settings.json)] コマンドは、ユーザー設定／ワークスペース設定の markdown-pdf.type 項目（後述）で指定されている出力形式への変換を行うものだ。

あるいは、エディタ（タブ）に表示している Markdown を右クリックして、コンテキストメニューから適切なものを選択してもよい。



コンテキストメニューから変換方法を選択してもよい

ファイルの保存と同時に HTML や PDF へ変換するには？

ここまで見てきたように、コマンドパレットやコンテキストメニューから明示的に Markdown を HTML や PDF に変換することもできるが、Markdown ファイルの保存時に自動的に変換を行うように設定することも可能だ。

これは、ユーザー設定あるいはワークスペース設定で指定する。設定する項目は次の 2 つだ。

- markdown-pdf.type 項目：出力フォーマットを指定（デフォルトは "pdf" のみ）
- markdown-pdf.convert onSave 項目：これを true にして、VS Code を再起動すると、Markdown ファイルの保存時に自動的に変換が行われるようになる

`markdown-pdf.type` 項目には、"pdf"（デフォルト）、"html"、"png"、"jpeg" を配列要素として指定していく。`markdown-pdf.convertOnSave` 項目には `true` か `false` のいずれかを指定する（デフォルト値は `false`）。以下に設定例を示す。

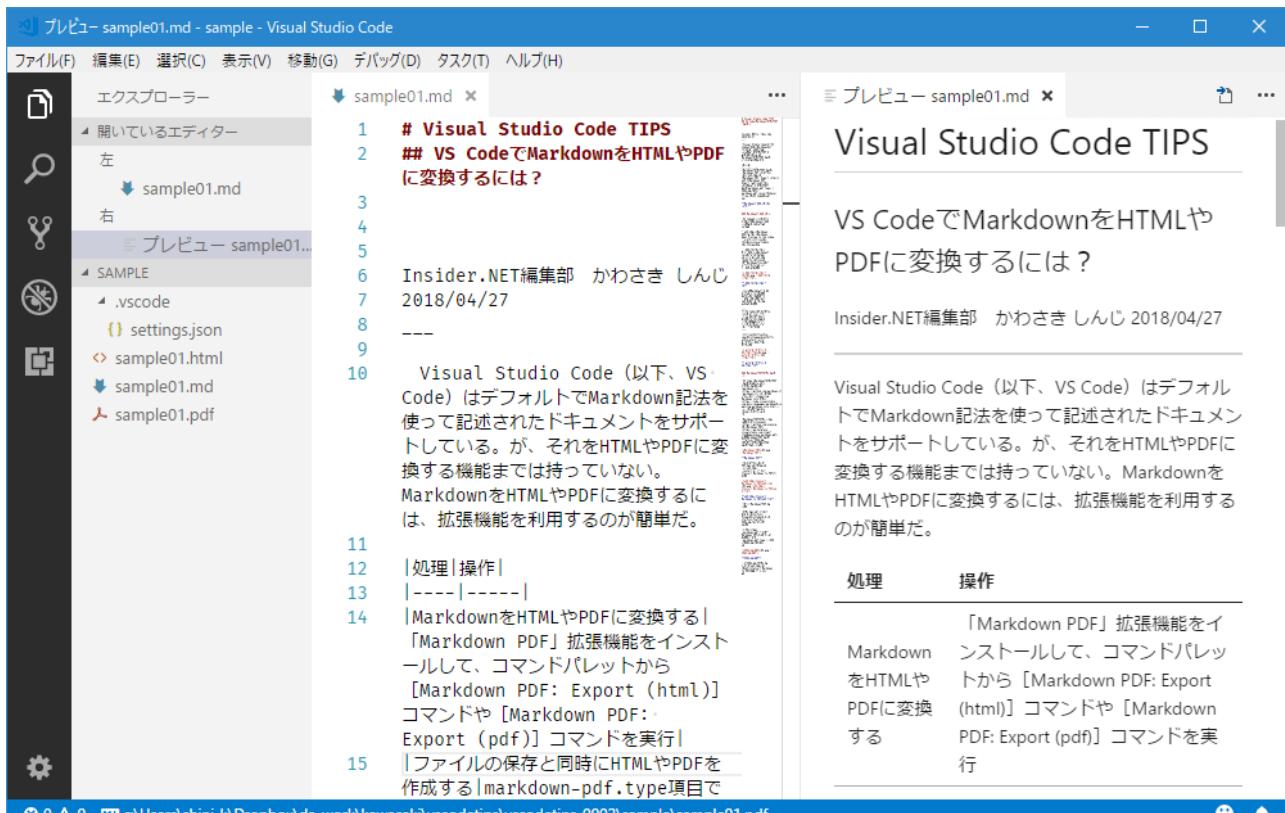
```
{  
  "markdown-pdf.type": [  
    "pdf",  
    "html"  
  ],  
  "markdown-pdf.convertOnSave": true  
}
```

`markdown-pdf.type` / `markdown-pdf.convertOnSave` 項目の設定例

この例では、変換先のフォーマットとして "pdf" と "html" を指定しているので、ファイルを保存すると、これらへの変換が自動的に行われる。なお、`markdown-pdf.type` 項目は先にも述べたが、[Markdown PDF: Export (settings.json)] コマンド実行時に作成するファイルの種類の指定でも使われている。

前述したが、`markdown-pdf.convertOnSave` の設定を有効にするには、VS Code の再起動が必要になるので注意しよう。

この設定を行って、Markdown ファイルを保存すると、変換が自動的に行われ、以下の画像のように、2つのファイルが作成される（[エクスプローラー] ビューに注目）。



`sample01.md` ファイルを保存すると、自動的に `sample01.html` ファイルと `sample01.pdf` ファイルが作成される

Markdown PDF 拡張機能の設定項目としては、この他にも以下のようなものがある（抜粋）。

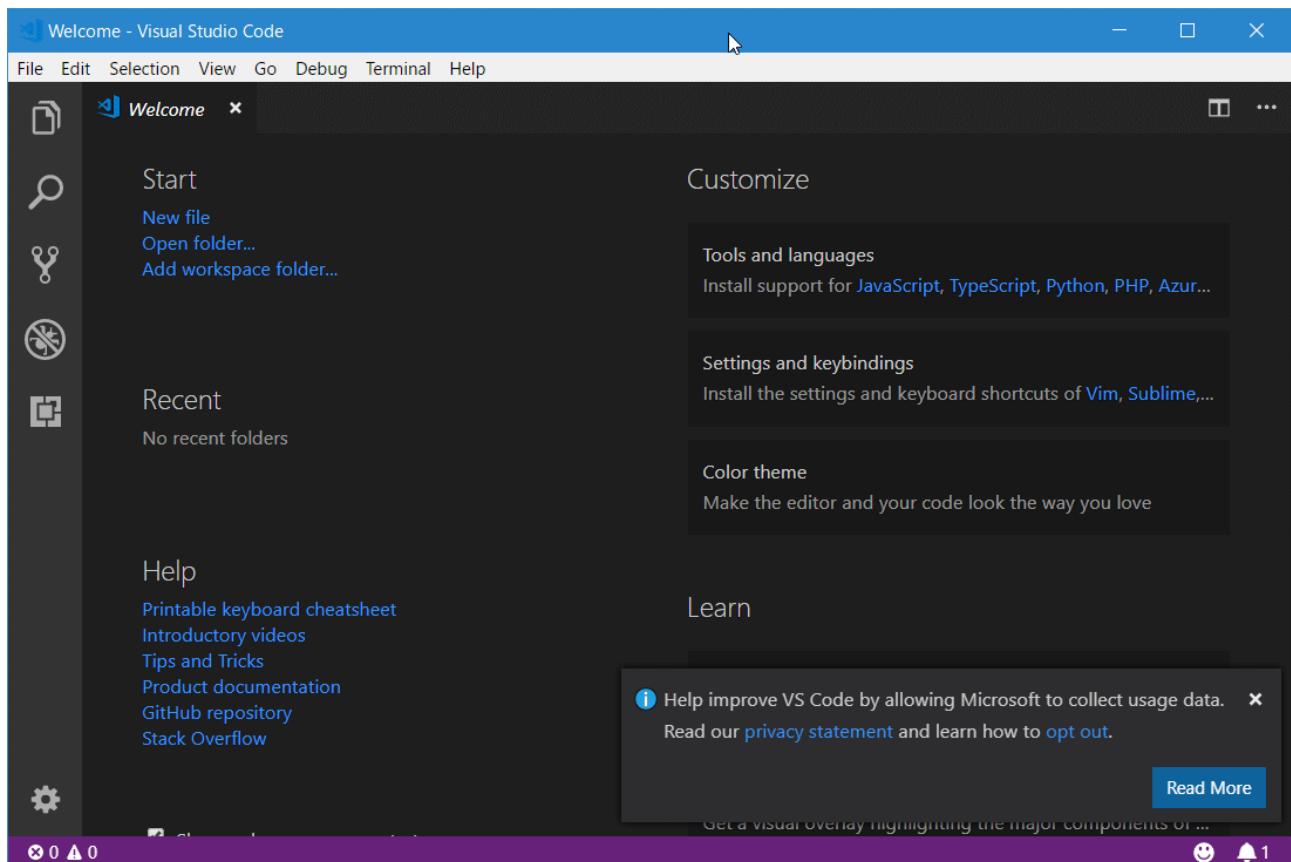
- ファイル保存時の自動変換で除外するファイルの指定
- 使用するスタイルシートの指定
- 構文ハイライト関連の設定
- ヘッダ／フッター関連の設定

これらの設定については、「[Markdown PDF](#)」ページを参照されたい。

VS Code で表示言語を変更するには

VS Code の GUI 表示で使われる表示言語を切り替える方法を解説。また、言語パック拡張機能のインストールについても取り上げる。

Visual Studio Code (以下、VS Code) はバージョン 1.25 からデフォルトの表示言語である英語だけが本体に同梱されるようになった。そのため、PC などに全く新規に VS Code をインストールすると以下のように英語を使って画面表示が行われるようになっている *1。



新規に VS Code をインストールしたところ

*1 VS Code をインストールしたことがある環境に、再インストールをした場合には、以前の設定が残っていることがあります。その場合には最初から日本語で UI 表示が行われるかもしれない（日本語言語パックが既にインストールされ、表示言語の構成が行われている場合）。

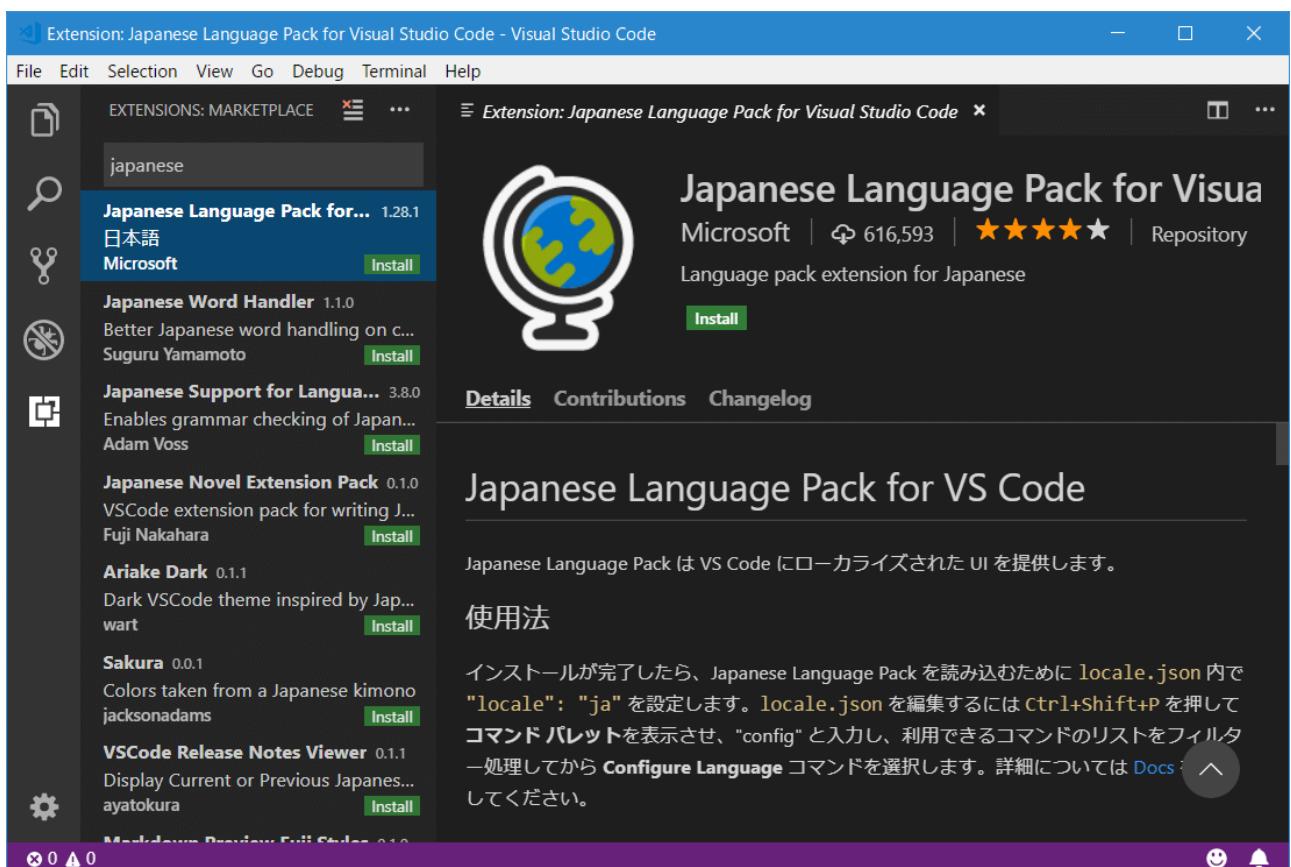
表示言語を（例えば）日本語にするには、日本語を表示するための言語パックのインストールと、表示言語の設定の 2 つを行う必要がある。また、言語パックをインストールしていると、VS Code 起動時にコマンドライン引数で指定して、その言語を表示言語として指定することも可能だ。

変更方法	操作
表示言語を定的に変更する	表示言語に対応する言語パック（拡張機能）をインストールして、 <code>locale.json</code> ファイルの <code>locale</code> 項目を変更する
表示言語を一時に変更する	<code>--locale</code> オプションを指定して、VS Codeを起動する

VS Code の表示言語の変更方法

定的に変更する

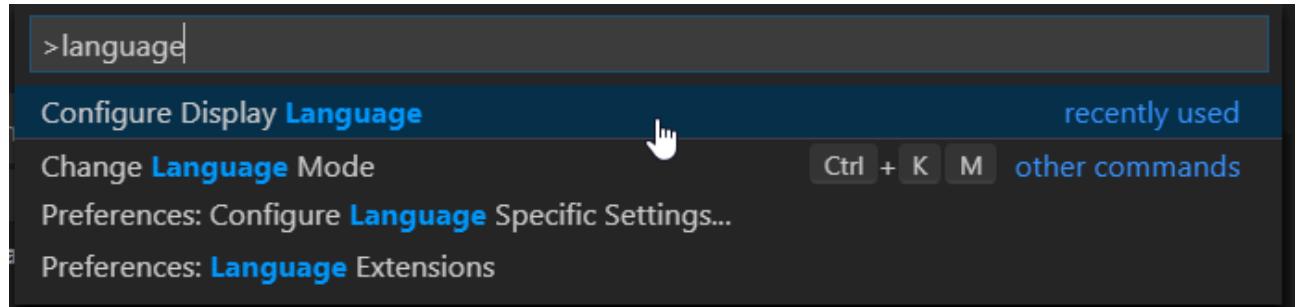
今も述べた通り、表示言語を英語以外の言語に変更するには、その言語用の言語パックが必要だ。言語パックは、VS Code の [EXTENSIONS] ビュー（[拡張機能] ビュー）からインストールできる。例えば、日本語用の言語パックをインストールするのであれば、[EXTENSIONS] ビューの一番上にある検索ボックスで「japanese」などを検索すればよい。



日本語用の言語パック（拡張機能）

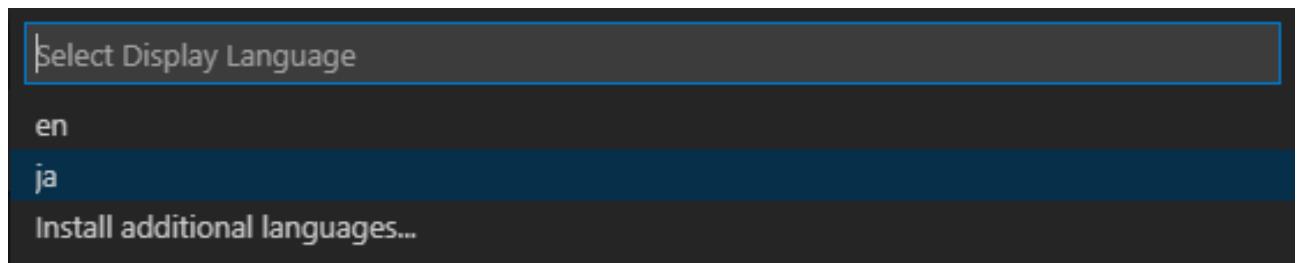
[Install] ボタンをクリックすると、言語パックのインストールが開始され、次に [Reload] ボタンが表示されたら、これをクリックして、VS Code を再起動すれば、言語パックのインストールは完了だ（バージョンによっては VS Code ウィンドウの再読み込みでもよい）。再起動により、インストールした言語パックを使って表示が行われるようになる。切り替わらない場合は以下の手順で表示言語を切り替えよう。

表示言語は明示的に変更できる。これには、[Ctrl] + [Shift] + [P] キー（Windows / Linux。macOS では [Shift] + [Command] + [P] キー）を押して、コマンドパレットを表示し、「config」や「language」などと入力してから [Configure Display Language]（[表示言語を構成する]）を選択する。



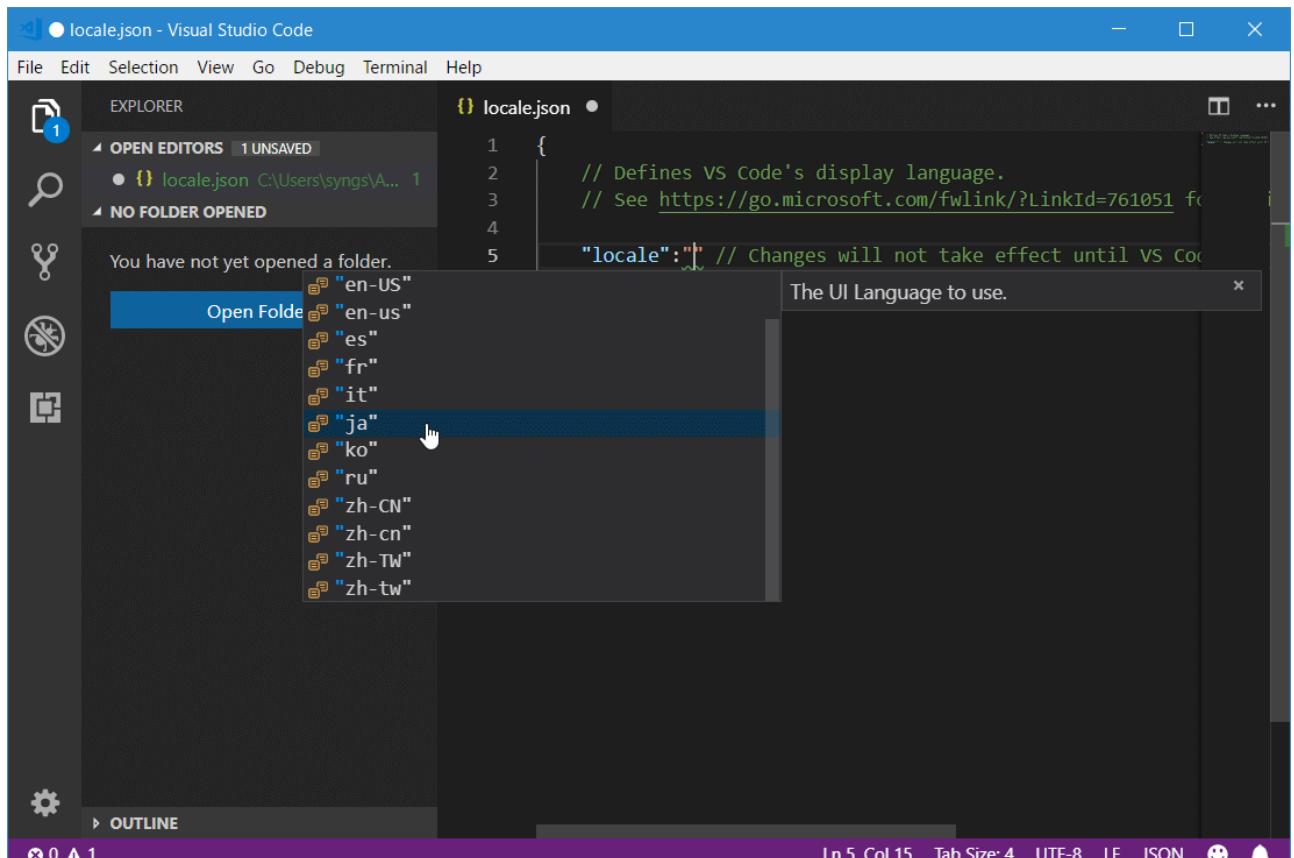
コマンドパレットから [言語を構成する] 項目を選択

これにより、VS Code 1.33 以降では、表示言語を選択するメニュー項目が表示される。ここから、使用する言語を選べばよい。日本語に切り替えるなら [ja] を選択する。



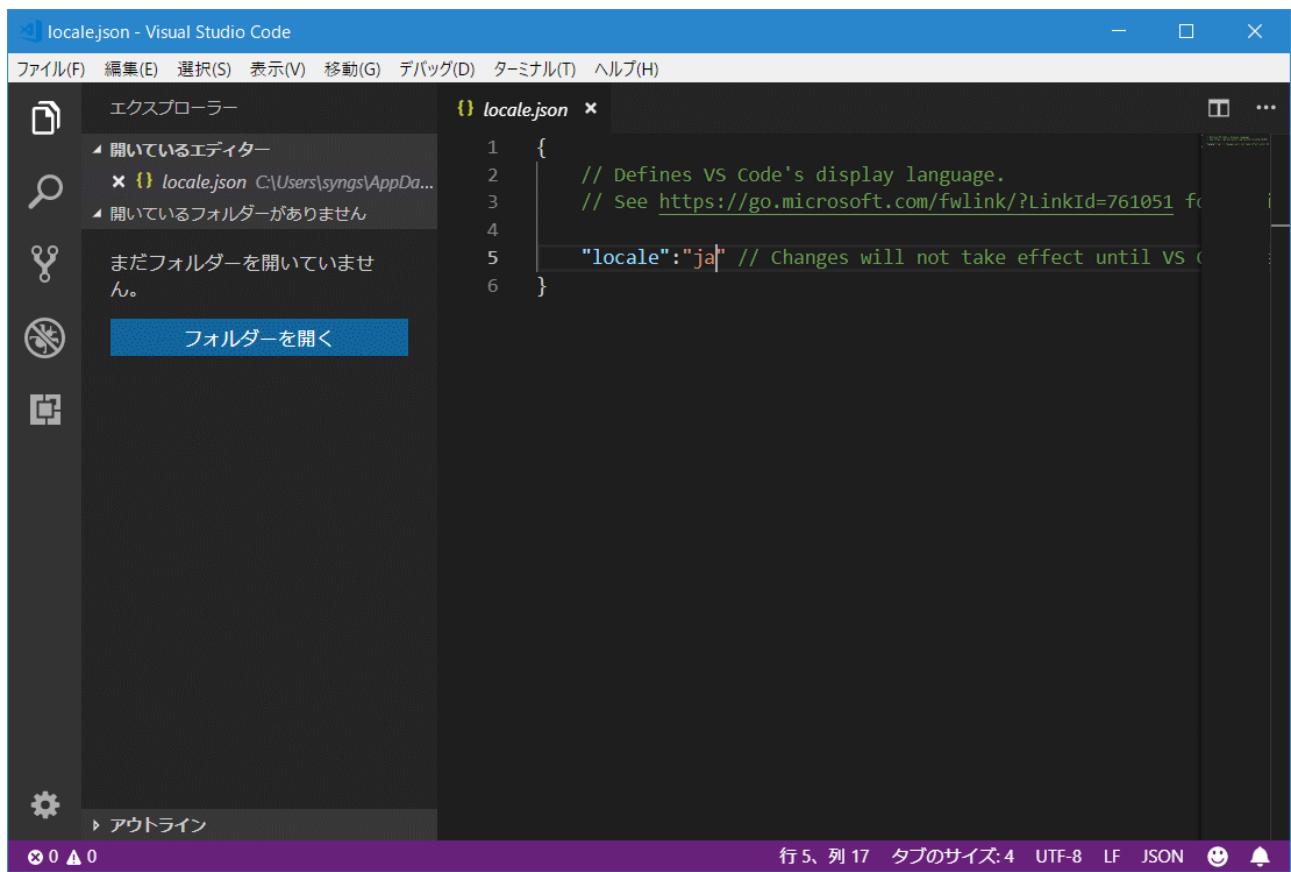
コマンドパレットから [言語を構成する] 項目を選択

以前のバージョンでは、エディタに locale.json ファイルが開かれるので、そこにある locale 項目に表示言語として使用する言語を指定すればよい。表示言語を日本語にするのであれば、これを "ja" にする。なお、locale.json ファイルはユーザー設定を保存する settings.json ファイルと同じ場所に保存される。例えば、Windows なら「%APPDATA%\Code\User\locale.json」 = 「C:\Users\<ユーザー名>\AppData\Roaming\Code\User\locale.json」などとなる。



locale 項目を設定しているところ

メニューから言語を選択した場合でも、`locale.json` ファイルを編集して保存した場合でも、その後に VS Code を再起動すれば、表示言語が切り替わる。



表示言語を日本語として、VS Code が再起動された

設定可能な値については「[Display Language](#)」を参照されたい（もちろん、対応する言語パックが必要になる）。

一時的に変更する

一時に VS Code の表示言語を切り替えるのであれば、コマンドライン（コマンドプロンプトや各種のシェル）から「`--locale`」オプションを指定して、VS Code を起動するのが簡単だ。ただし、切り替え先の言語に対応した言語パックが必要になる（英語以外）。

VS Code を起動する「`code`」に続けて、「`--locale <表示言語のロケール>`」を指定してやればよい。例えば、英語を表示言語とするのであれば、「`code --locale en`」などとなる。

VS Code の古いバージョンではインストールすればすぐに日本語で GUI が表示されていたが、VS Code 1.25 以降では、デフォルトの表示言語である英語以外の言語で GUI を表示するには、言語パックのインストールと表示言語の構成が必要になる。普段使いのマシンでは気にすることもないかもしれないが、PC の切り替えなどで新規に VS Code をインストールするときには注意しよう。

VS Code を持ち運ぶには(ポータブルモード)

VS Code 1.25 以降でサポートされている「ポータブルモード」で、普段とは別の環境でも、いつもの設定で VS Code を使えるようにしてみよう。

Visual Studio Code (以下、VS Code) はバージョン 1.25 から「ポータブルモード」をサポートするようになった。この機能を利用して、自分が普段使っている設定のまま、VS Code を USB メモリやファイル共有を利用して、他のマシンでも実行する方法を説明する。

VS Code を持ち運ぶための手順

1. VS Code の[ダウンロードページ](#)から、Windows では ZIP 形式、Linux では .tar.gz 形式で配布されている VS Code を入手する。macOS では ZIP 形式で配布されている通常のパッケージを入手する
2. ZIP ファイル／.tar.gz ファイルを（任意のフォルダに）展開する
3. 設定ファイルや拡張機能を保存する「data フォルダ」を作成する
4. data フォルダに、自分の環境を復元するのに必要なファイルをコピーする
5. ポータブルモードの VS Code が入ったフォルダを USB メモリやファイル共有に配置する

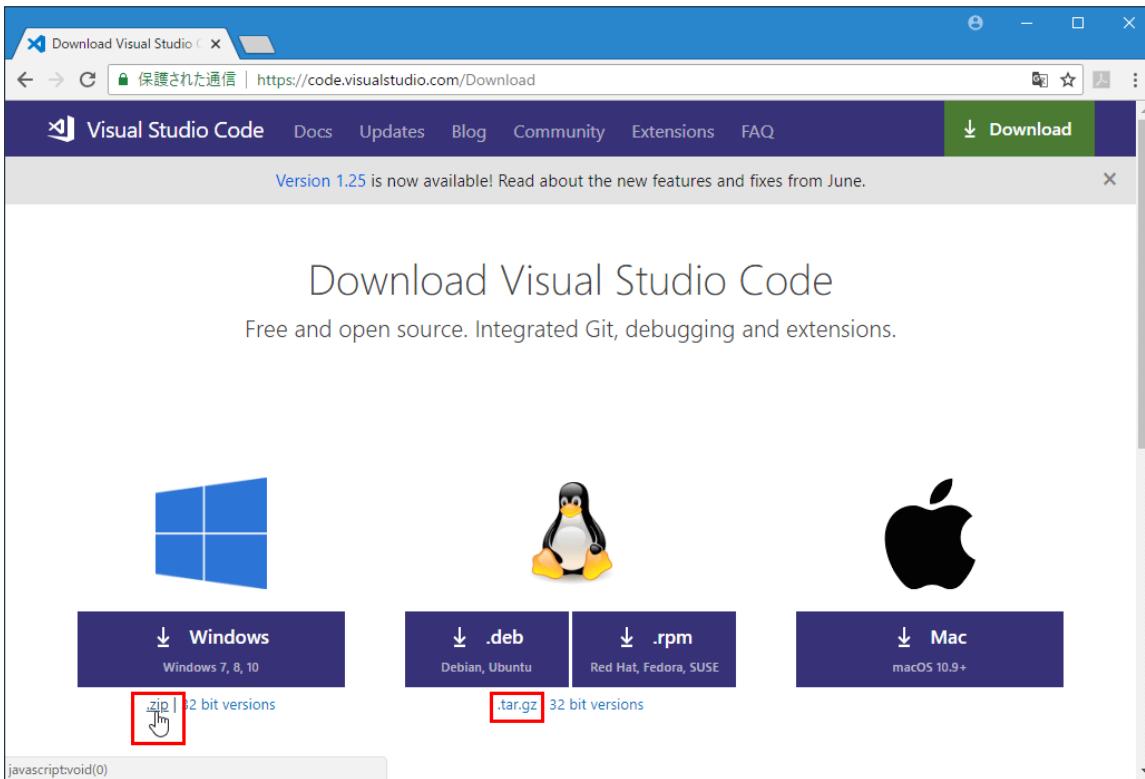
ポータブルモードとは

「ポータブルモード」とは、VS Code の実行ファイルを含むトップレベルのフォルダ以下（と、macOS ではトップレベルのフォルダと同じレベルにある data フォルダ）に、その実行に必要な全てのファイルを含んだ VS Code のこと。

ポータブルモードの VS Code をセットアップしておけば、そのフォルダを USB メモリにコピーしたり、ファイル共有を利用したりするだけで、VS Code をインストールしていない環境でもそこから「普段、自分が使っている構成の VS Code」を実行できる。他者の環境でちょっとした作業をしなければならないといったときに、USB メモリに普段使いの VS Codeを入れておくだけで、いつでもいつもの使い勝手で作業できるのは少しうれしい。

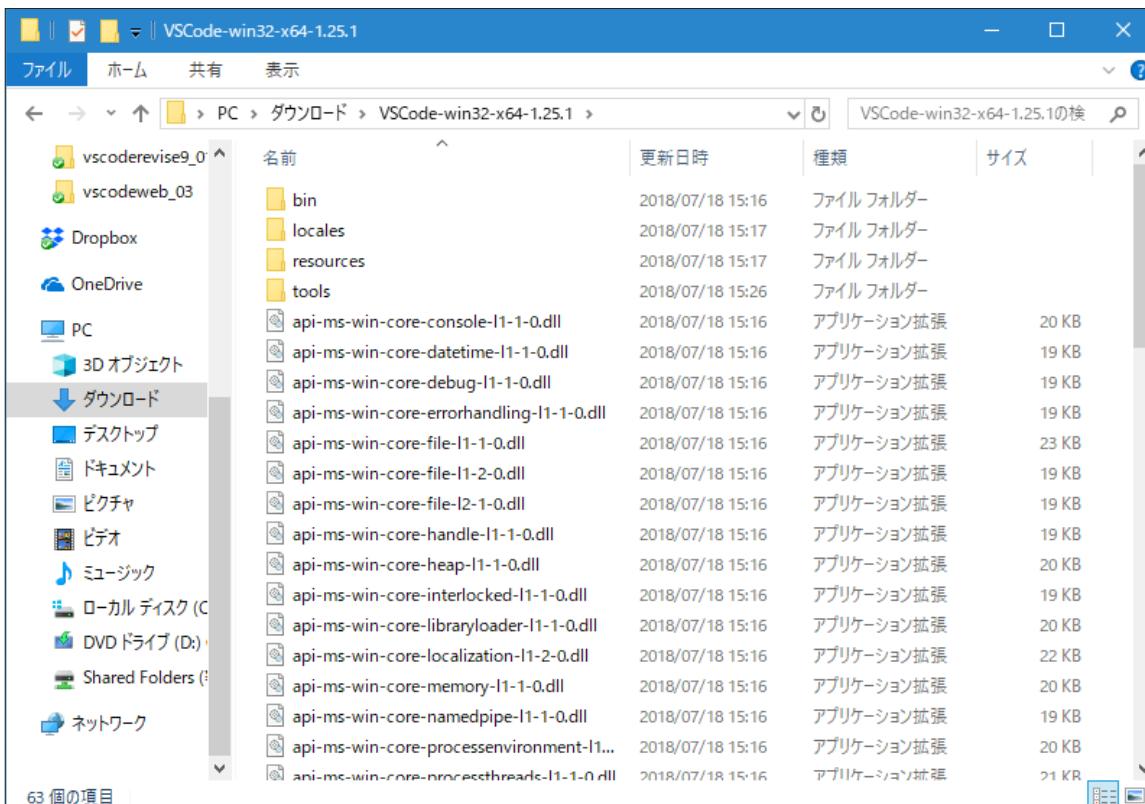
ポータブルモードの VS Code をセットアップする

ポータブルモードの VS Code をセットアップするには、まず ZIP ファイルとして配布されている VS Code を手に入れる（Linux では .tar.gz ファイルをダウンロードする。macOS は ZIP 形式のパッケージのみが配布されているので、いつも通りにダウンロードするだけよい）。



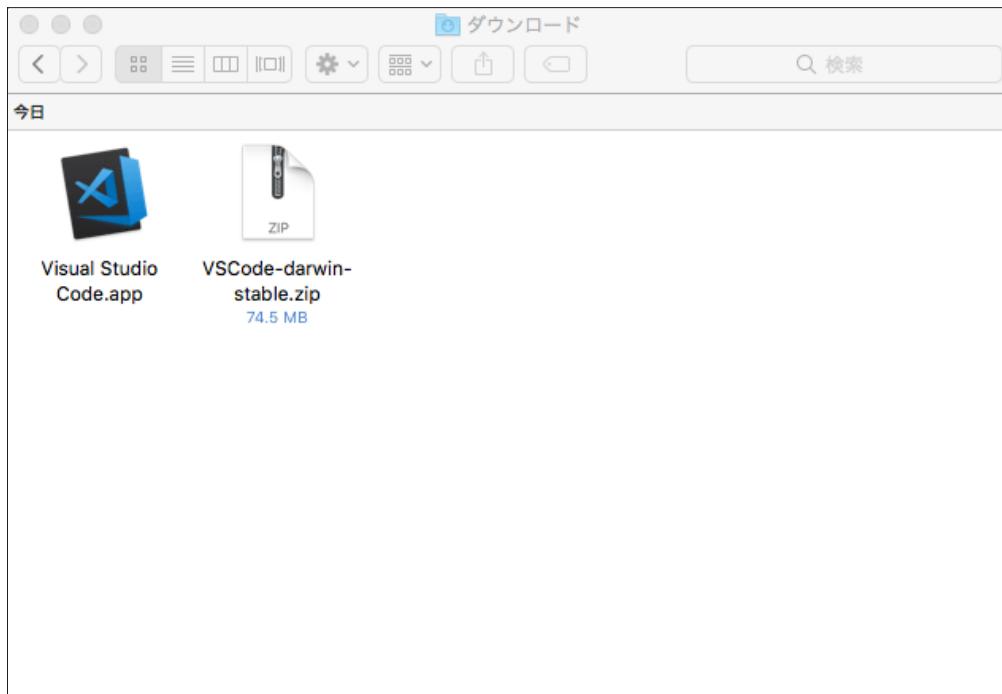
Windows / Linux では、VS Code のダウンロードページにある [.zip] あるいは [.tar.gz] のリンクをクリック（赤枠内）

ZIP ファイル（もしくは .tar.gz ファイル）をダウンロードしたら、これを展開する。Windows 版であれば、次のように多くのファイルが目に見える形で展開される（Linux も同様）。Windows ではあらかじめ「[ロックを解除](#)」しておこう。



Windows 用の ZIP ファイルを開いたところ

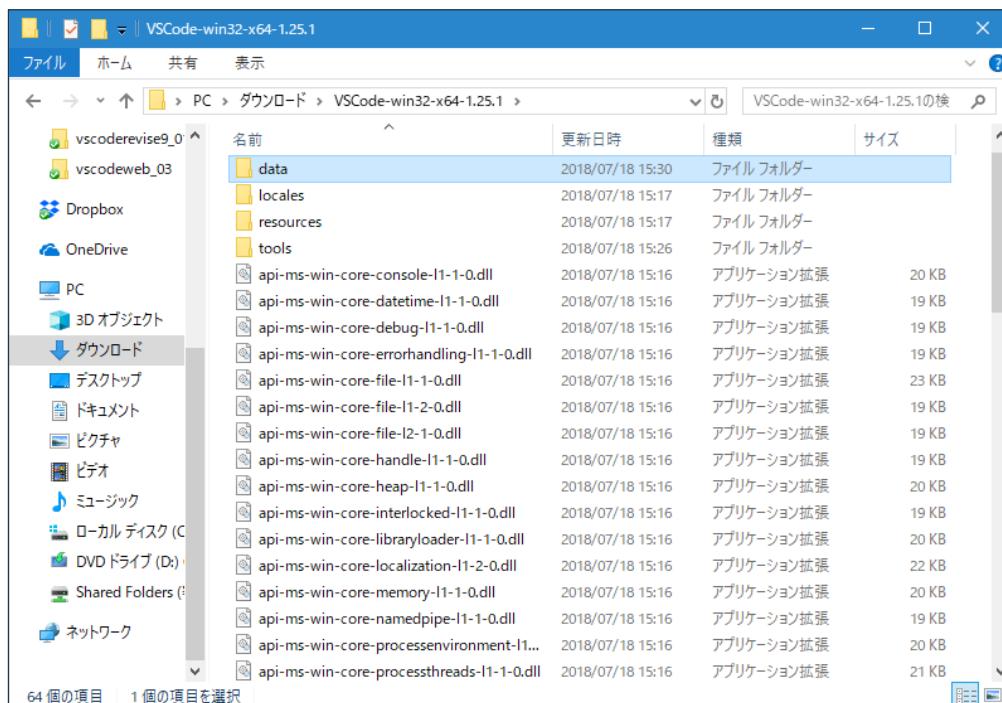
macOS の Finder では「Visual Studio Code.app」ファイルが展開される。



macOS 用の ZIP ファイルを展開したところ

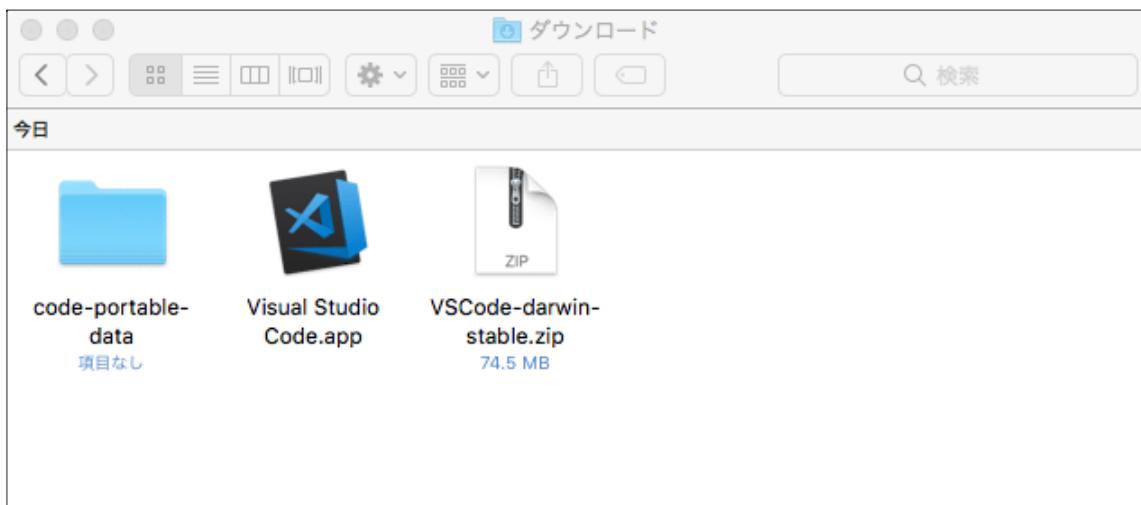
macOS では、この後でコンソールから「xattr -dr com.apple.quarantine Visual Studio Code.app」コマンドを実行しておく（これを実行しないとポータブルモードで動作しない。あるいは起動しない）。

ダウンロードと展開が済んだら、Windows / Linux では、展開先のフォルダに「data」という名前のフォルダを作成する。



data フォルダを作成

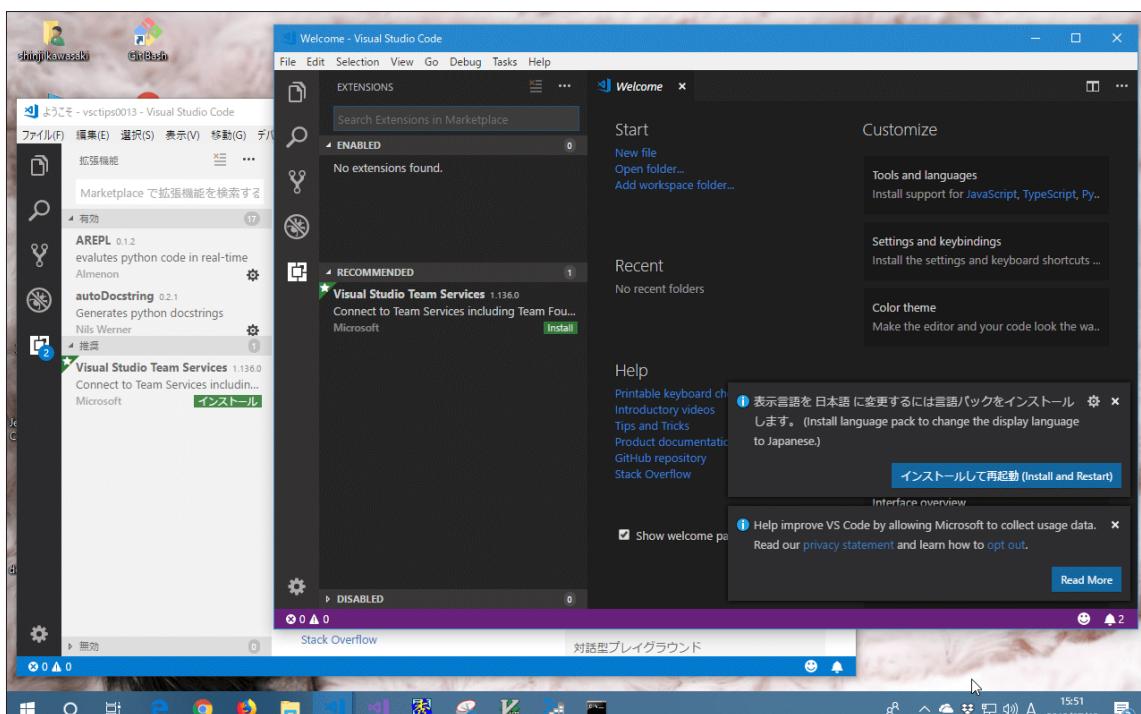
macOS では、展開先のフォルダと同じレベルのフォルダに「code-portable-data」という名前のフォルダを作成する。



code-portable-data フォルダを作成

data フォルダ（code-portable-data フォルダ）には、VS Code のユーザーデータ（セッション情報や各種キャッシュファイルなど）や拡張機能など、VS Code を使っている上で作成された情報やカスタマイズした情報を保存しておくためのもの。Windows / Linux では展開先のフォルダ内にこれを置くが、macOS では「Visual Studio Code.app」ファイルがあるフォルダに置く必要があることと、「code-portable-data」という名前になることには注意しよう。以下では、このフォルダを data フォルダとして記述していく。

data フォルダが空のままで、ポータブルモードの VS Code を起動すると、次のようになる。



ポータブルモードの VS Code と、通常の VS Code を起動したところ

手前の黒いバックグラウンドのウィンドウがポータブルモードの VS Code で、奥の白いバックグラウンドのウィンドウが通常の VS Code だ（筆者の環境）。data フォルダには何も設定情報を含めていないので、ポータブルモードの VS Code は英語表示で背景が黒くなっている（デフォルトの「Default Dark+」テーマが使われている）。両者の【拡張機能】ビューを見ると分かるが、ポータブルモードには拡張機能も入っていない（そのため、日本語をサポートする言語パック拡張機能のインストールが推奨されている）。

そこで、自分が普段使用している VS Code のユーザーデータや拡張機能を（必要に応じて）data フォルダにコピーしていく必要がある。

ユーザーデータと拡張機能のコピー

まずユーザーデータは、OS ごとに以下のフォルダとなっているので、これらを data フォルダにコピーして、フォルダ名を「user-data」に変更する。

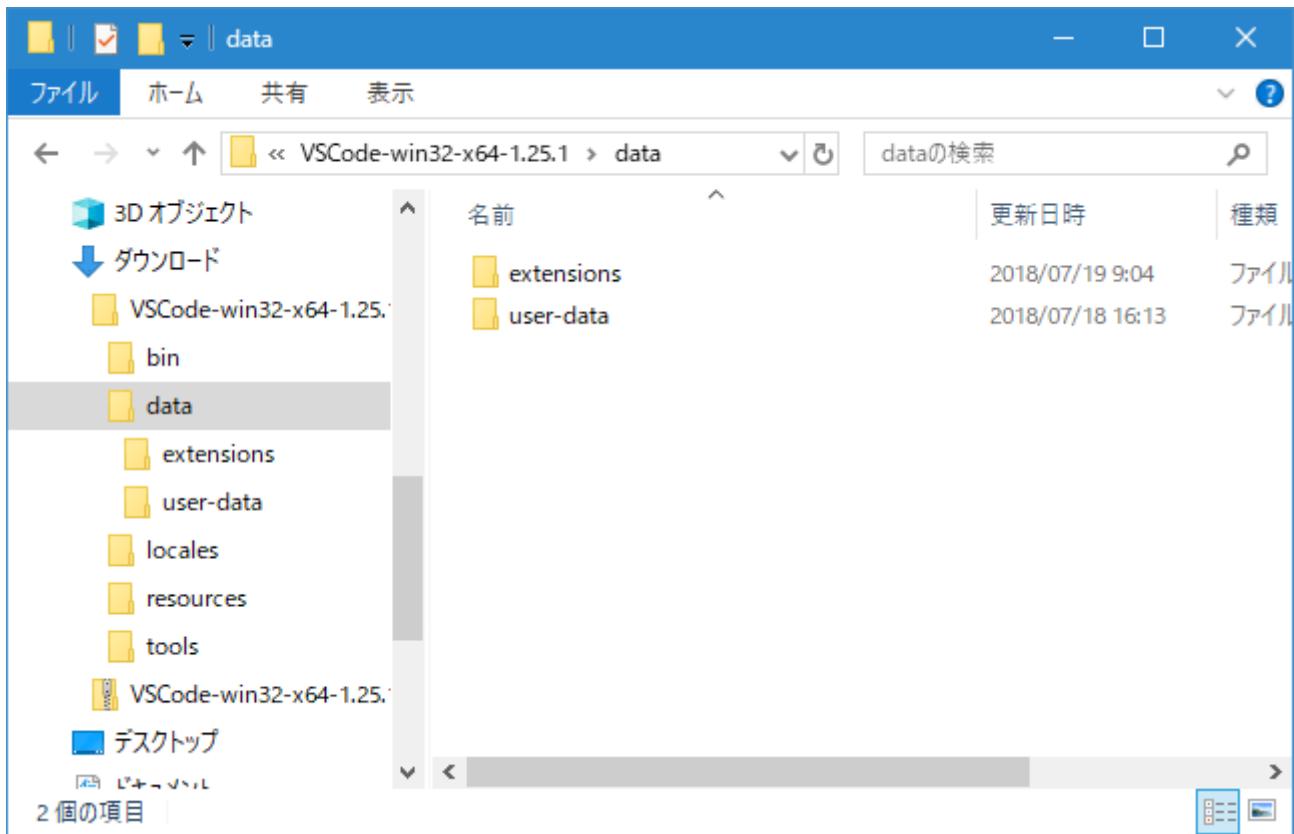
- Windows : %APPDATA%\Code フォルダ
例：「C:\Users\< ユーザー名 >\AppData\Roaming\Code」 フォルダ
- Linux : \$HOME/.config/Code フォルダ
例：「/home/< ユーザー名 >/config/Code」 フォルダ
- macOS : \$HOME/Library/Application Support/Code フォルダ
例：「/Users/< ユーザー名 >/Library/Application Support/Code」 フォルダ

VS Code にインストール済みの拡張機能は上記フォルダとは別の場所にあるので、以下のフォルダを同じく data フォルダにコピーする（こちらはフォルダ名を変更する必要はない）。

- Windows : %USERPROFILE%\.vscode\extensions フォルダ
例：「C:\Users\< ユーザー名 >\.vscode\extensions」 フォルダ
- Linux : ~/.vscode/extensions フォルダ
例：「/home/< ユーザー名 >/.vscode/extensions」 フォルダ
- macOS : ~/.vscode/extensions フォルダ
例：「/Users/< ユーザー名 >/.vscode/extensions」 フォルダ

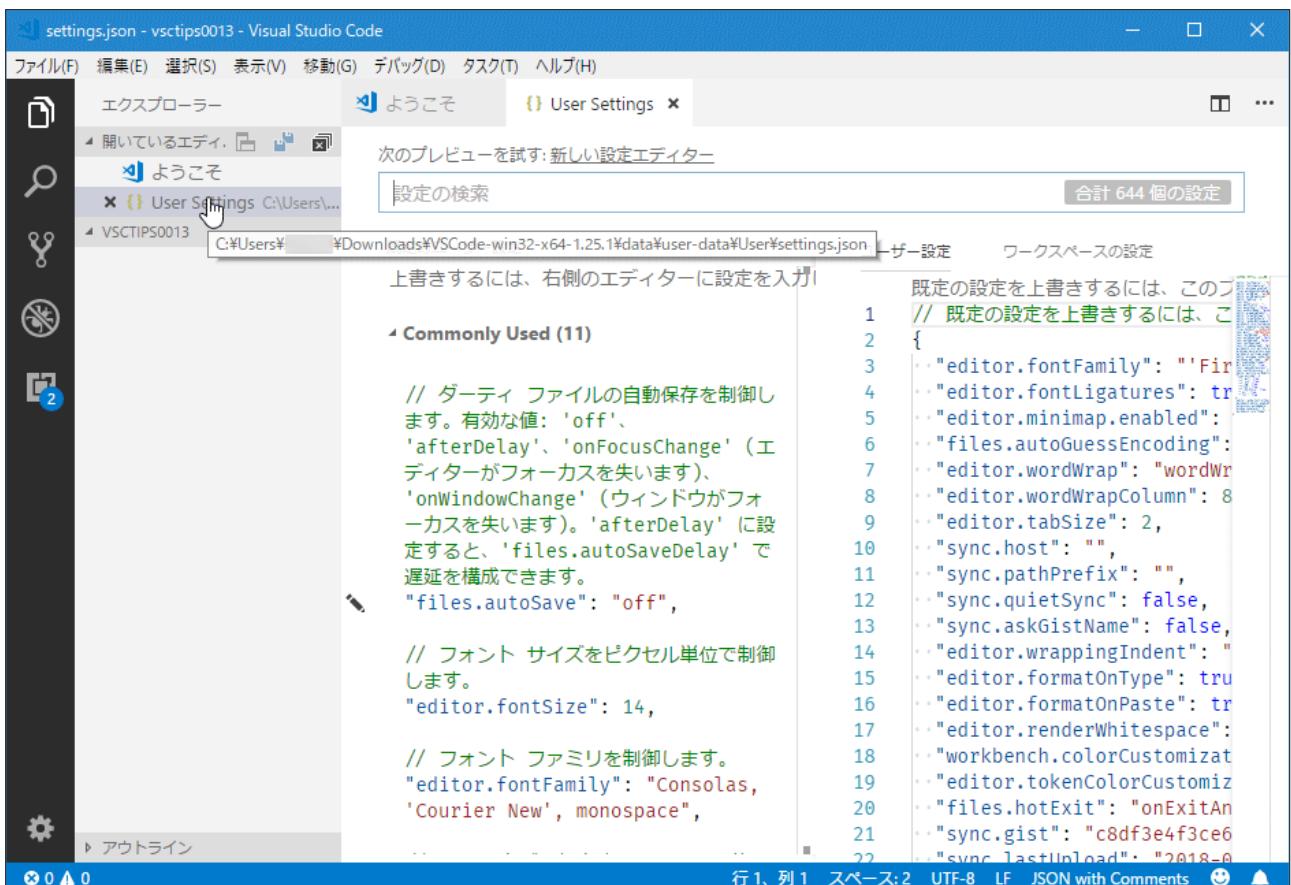
インストールしている拡張機能の数にもよるが、コピーには時間がかかるかもしれない。

結果、data フォルダには「user-data」と「extensions」の 2 つのフォルダがあり、その下にそれぞれユーザーデータと拡張機能が含まれるようになる。以下に、入手した ZIP ファイルを「ダウンロード」フォルダで展開して、data フォルダを作成し、そこに上記の 2 つのフォルダを追加したところを示す（Windows 版の例）。



data フォルダの内容

以上で、ポータブルモードの VS Code に普段使いの VS Code のユーザーデータと拡張機能が追加された。この状態で、ポータブルモードの VS Code を起動して、ユーザー設定の編集画面を表示したところを以下に示す。



ポータブルモードの VS Code でユーザー設定を編集しているところ

下の画像は settings.json ファイルのパスにズームしたところ。通常の settings.json ファイルのパスとは異なっているところに注目されたい。

[エクスプローラー] ビューでポップアップされているファイルパスを見ると分かる通り、通常の異なる位置（「ダウンロード」フォルダに展開したポータブルモードの VS Code の user-data フォルダのサブフォルダ）にある settings.json ファイルが開かれている。このことから想像できるが、ポータブルモードの VS Code では、data\user-data\User フォルダにユーザー設定を記述する settings.json ファイル、キーボードショートカットを記述する keybindings.json ファイル、表示言語を設定する locale.json ファイルが保存されている。ポータブルモードで設定を変更すると、これらのファイルにその情報が書き込まれるようになる。

なお、`data` フォルダは「ポータブル」だ。つまり、VS Code のバージョンが上がったときには、新バージョンの ZIP ファイル (`.tar.gz` ファイル) をダウンロード／展開して、以前の `data` フォルダをコピーするだけで、新バージョンのポータブル版 VS Code の設定が完了する。とはいえ、普段使いの VS Code では設定を変更したり、拡張機能をインストールしたりといったことが（ある程度）頻繁には行われるであろうから、2 つの VS Code の間で設定の同期を取るのは面倒な作業となるかもしれない。`macOS` では自動的にアップデートしてくれるようだ。

後は、これを USB メモリにコピーしたり、ファイル共有でアクセスできるようにしたり、あるいは `Dropbox` などのサービスに配置したりしておけば、自分の好みに設定が完了している VS Code をいつでも使えるようになるはずだ。

「ポータブルモード」や「ポータブルアプリ」といった語は日本ではあまりなじみがないかもしれないが、[PortableApps.com](#) など、「ポータブル」なアプリを集めた著名なサイトもある。VS Code 以外にも、Web ブラウザのポータブル版など、必要最小限のアプリをこうしたサイトから入手して、まとめておくと、普段とは異なる環境で作業するときに役立つだろう。

VS Code でウィンドウサイズを制御するには

`window.newWindowDimensions` 項目の値を設定することで、VS Code のウィンドウを開くときのサイズをある程度制御することが可能だ。

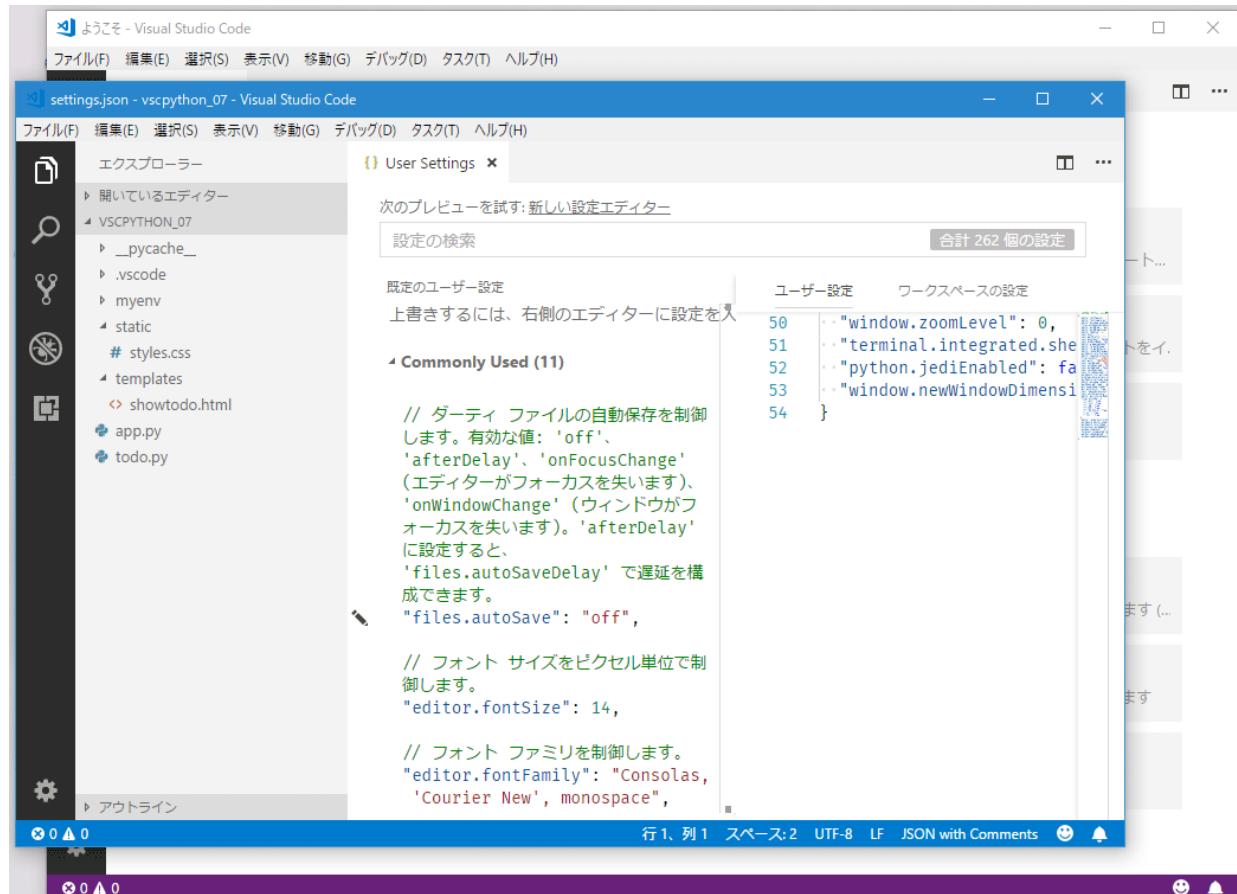
Visual Studio Code (以下、VS Code) でウィンドウを開いたときの挙動には一定のルールがある。本稿では、VS Code のウィンドウを複数開くときに、そのサイズを制御する方法を説明する。

VS Codeのウィンドウを開くときのサイズ	window.newWindowDimensions項目に設定する値
画面中央に一定のサイズでウィンドウを開く	"default"
直前にアクティブなVS Codeのウィンドウと同じサイズでウィンドウを開く	"inherit"
最大サイズでウィンドウを開く	"maximized"
全画面表示でウィンドウを開く	"fullscreen"

VS Code のウィンドウサイズを制御する

VS Code でウィンドウを開くときには、次のような挙動になる（デフォルト）。

- 1つ目のウィンドウは、前回に閉じたウィンドウと同じサイズ、同じ位置に開かれる
- それ以降のウィンドウは、画面中央に一定のサイズで開かれる



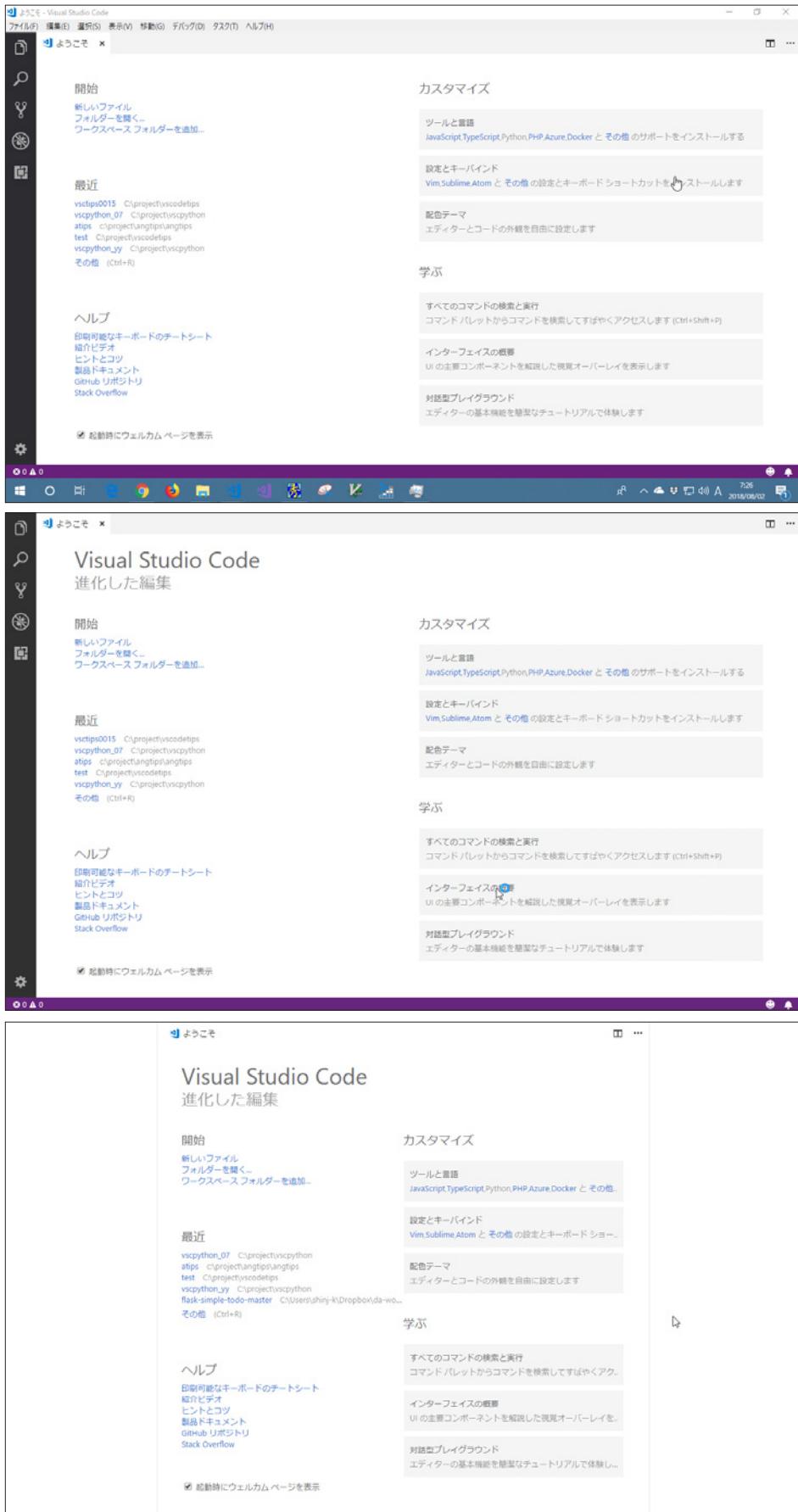
手前のウィンドウは筆者が普段使っているサイズのウィンドウで、奥のウィンドウは VS Code が自動的に位置とサイズを決定したもの

ウィンドウサイズをユーザーが事細かに指定して制御することはできないが、ユーザー設定で `window.newWindowDimensions` 項目を設定することで、「2つ目以降のウィンドウのサイズ」をある程度制御することは可能だ（1つ目のウィンドウは常に上に示した方法で開かれる）。なお、これはユーザー設定でのみ扱える項目だ（とはいえ、ワークスペース設定でこれを変更する意味はないはずだ）。設定可能な値は以下の4つとなっている。

- "default"：画面中央に一定のサイズで新規ウィンドウを開く
- "inherit"：直前にアクティブだった VS Code ウィンドウと同じサイズで新規ウィンドウを開く
- "maximized"：最大化状態で新規ウィンドウを開く
- "fullscreen"：全画面を占拠する形で新規ウィンドウを開く

"default" を設定した際の動作は既に述べた通りだ。"inherit" は直前にアクティブだった VS Code ウィンドウのサイズを使って開かれる。

"maximized" と "fullscreen" は共に画面を覆うようにウィンドウを開くが、その違いは OS によって異なる。例えば、Windows 版でこの項目の値を "maximized" に設定すると、ウィンドウは「タイトルバーとメニューバー付き」で最大化表示される。"fullscreen" に設定すると、ウィンドウは画面全体を覆うように表示され、タイトルバーとメニューバーもなくなる（Windows の場合、画面下部のタスクバーも表示されなくなる）。ちなみに、コーディングに集中するための「Zen Mode」では VS Code のサイドバーとステータスバーもなくなるが、これとはまた違った形だ。以下に例を示す。これらは全て、筆者の作業環境の画面を全画面キャプチャーしたものだ。



"maximized" と "fullscreen" の違い

上は `window.newWindowDimensions` 項目の値を "maximized" にした場合の画像。中は "fullscreen" にした場合の画像。下は「Zen Mode」の画像（参考用）。

macOS ではメニューバー領域を除く部分でウィンドウを最大化するか、画面を完全に VS Code ウィンドウが占拠するかの違いとなる（Linux でもほぼ同様）。

オススメの設定値は "inherit" だ。既に書いた通り、これは「直前にアクティブだったウィンドウと同じサイズ」でウィンドウを開くもの。そのため、実質的には次のような使いができる。

1. 自分好みのサイズにしてあった VS Code を閉じて終了する
2. 次回の VS Code 起動時には、そのサイズでウィンドウが開かれる
3. その後、ウィンドウを開くたびに（ウィンドウサイズを変更していなければ）2. で開いたサイズで開かれる
4. 1. に戻る

というわけで、「VS Code はいつもこのサイズで使いたい」と考えているのであれば、`window.newWindowDimensions` 項目の値を "inherit" にしておくのがオススメだ。

VS Code でウィンドウサイズを制御するには

VS Code で空白文字（スペース）を一目で分かるように表示する方法と、拡張機能を使って全角の空白文字を可視化する方法を紹介。

プログラムやドキュメントを記述しているときには、空白文字はまさに「空白」のように見えるが、何らかの体裁で空白文字を描画（表示）することで「ここに空白文字が含まれている」と一目で分かり、助かる（あるいは精神衛生的によい）ことがある。Visual Studio Code（以下、VS Code）では、`editor.renderWhitespace` 項目を設定することで、ファイル中に含まれている空白文字の描画方法（表示／非表示）を切り替えられる。

操作	
空白文字の描画方法を切り替える	<code>editor.renderWhitespace</code> 項目を設定する。設定可能な項目は以下の通り "none" : 空白文字を描画しない "boundary" : 単語を分ける单一の空白文字以外は描画する "all" : 空白文字を全て描画する
全角の空白文字を描画する	<code>zenkaku</code> 拡張機能を利用する

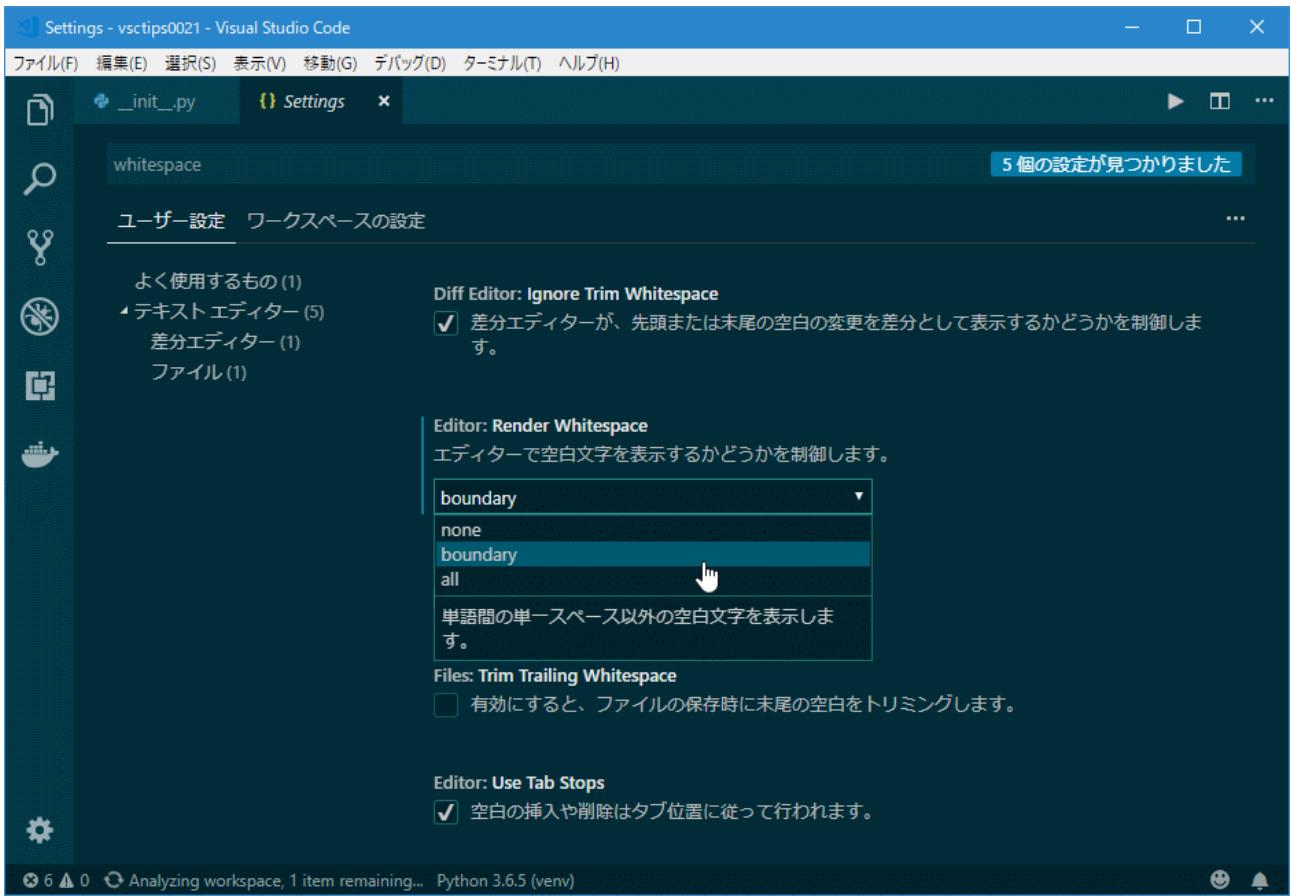
空白文字の描画方法を切り替える方法

空白文字の描画方法を切り替える

VS Code ではユーザー設定／ワークスペース設定で `editor.renderWhitespace` 項目を設定することで、空白文字の描画方法（表示形式）を切り替えられる。設定可能な値は次の 3 つだ。

- "none" : 空白文字を描画しない（空白のまま）
- "boundary" : 単語を分ける单一の空白文字以外は描画する
- "all" : 全ての空白文字を描画する

VS Code 1.27 で正式導入された新しい設定エディタでも、それ以前の設定エディタでも「whitespace」などを検索すれば、該当する項目が表示されるので、好みのものを設定すればよい。以下は新しい設定エディタで、この項目を設定しているところだ。



新しい設定エディタで editor.renderWhitespace 項目を設定しているところ

設定可能な値の中で分かりづらいのは "boundary" だが、これは「abc def」のように単語やキーワード、識別子などを区切る空白文字が 1 つだけある場合には、それを描画せず空白のままとして、「abc def」のように空白文字が 2 つ以上連続している場合には、「・」を使って描画するというものだ。言葉にすると分かりにくいので、以下に 3 つの設定値で空白文字の表示がどうなるかを示す。

The image consists of three vertically stacked screenshots of the Visual Studio Code interface, all displaying the same Python code in a file named `_init_.py`.

```

4  from flask_cors import CORS
5  import os
6  from todoapi.config import Config, DevConfig, DockerConfig
7
8  config = {
9      "default": Config,
10     "dev": DevConfig,
11     "docker": DockerConfig
12 }
13
14 app = Flask(__name__, instance_relative_config=True)
15
16 cfg = config[os.getenv("TODOAPI_ENV", default="default")]

```

The code is identical in all three instances, but the rendering of whitespace (spaces and tabs) differs based on the `editor.renderWhitespace` setting:

- Top Screenshot (None):** Shows standard rendering where spaces and tabs are preserved as they appear in the code.
- Middle Screenshot (boundary):** Shows rendering where individual spaces and tabs are preserved, but tab characters are represented by their ASCII value (e.g., '\t' instead of a visual tab).
- Bottom Screenshot (all):** Shows rendering where all whitespace characters (spaces and tabs) are converted into a single character, specifically a dot ('.') in this context.

editor.renderWhitespace 項目の値による空白文字の描画方法の違い

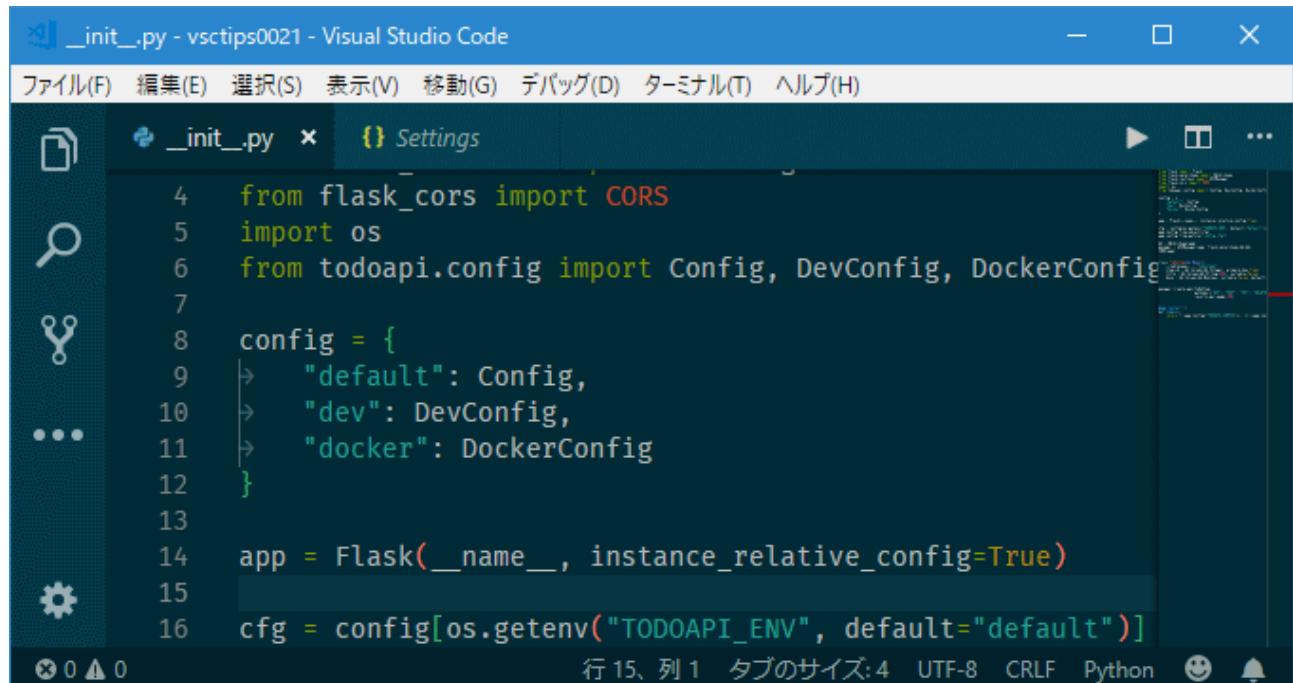
上："none" に設定した場合。空白文字は「空白」のまま。

中："boundary" に設定した場合。単独の空白文字は「空白」のまま。連続する場合は「・」を用いて描画される。

下："all" に設定した場合。空白文字が全て「・」で描画される。

「config = ……」以降の行に注目すると、各設定値でどのような表示になるかが分かりやすい。特に "boundary" の例（中）では、インデントを付ける空白文字は「・」を使って描画されているが、「"default": Config」に含まれる空白文字は描画されていない（空白文字が 1 つだけなので）。一方、「"docker": DockerConfig」には 2 つの空白文字が連続しているため、それらが描画されるようになっている。

また、タブ文字の描画方法もこの項目の設定で切り替えられる。以下に例を示す。



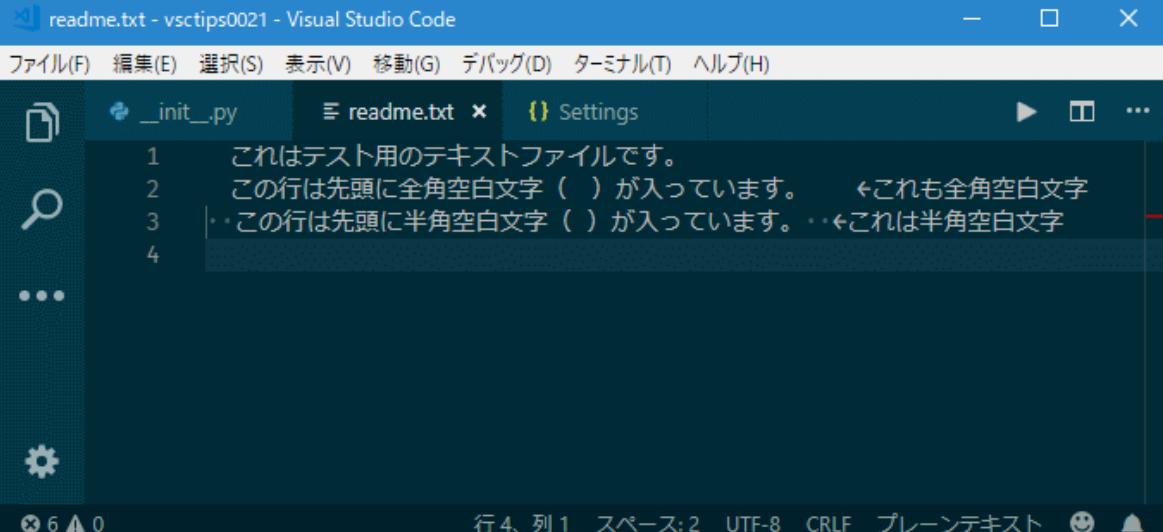
```
4     from flask_cors import CORS
5     import os
6     from todoapi.config import Config, DevConfig, DockerConfig
7
8     config = {
9         "default": Config,
10        "dev": DevConfig,
11        "docker": DockerConfig
12    }
13
14    app = Flask(__name__, instance_relative_config=True)
15
16    cfg = config[os.getenv("TODOAPI_ENV", default="default")]
```

タブ文字でインデントを付けるようにしたコード

上と同様に「config = ……」以降の行に注目してみよう。上の例では、インデントをタブ文字で付けるように設定を変更しているが、インデントを付けた部分には「→」を使ってタブ文字が描画されているのが分かるはずだ。また、ここでは示さなかったが、タブと半角空白文字が混在していれば、それぞれが異なる体裁で描画されるので、一目で視認できる。

全角の空白文字を描画する

editor.renderWhitespace 項目では、全角の空白文字の描画方法までは切り替えられない。以下に例を示す。



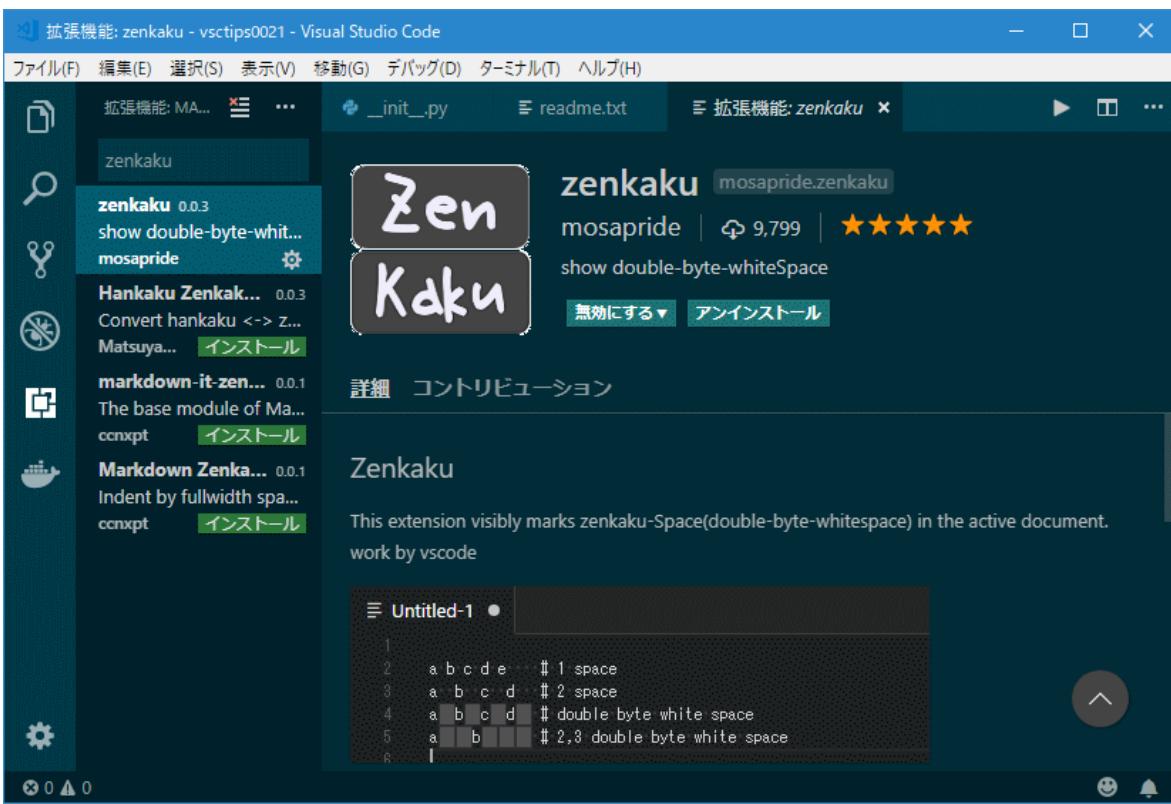
```
 readme.txt - vsctips0021 - Visual Studio Code
 ファイル(F) 編集(E) 選択(S) 表示(V) 移動(G) デバッグ(D) ターミナル(T) ヘルプ(H)
 _init_.py readme.txt Settings
 1 これはテスト用のテキストファイルです。
 2 この行は先頭に全角空白文字（　）が入っています。←これも全角空白文字
 3 |・この行は先頭に半角空白文字（　）が入っています。・←これは半角空白文字
 4

 行 4、列 1 スペース: 2 UTF-8 CRLF プレーンテキスト ☺ 🔔
```

全角の空白文字は描画されない

上の例に示したテキストファイルには、全角と半角で空白文字を入れてあるが、全角の空白文字は描画されていないことが分かる（editor.renderWhitespace 項目を "boundary" に設定しているので、単独の半角空白文字は描画されていない）。

全角の空白文字を描画するには、zenkaku 拡張機能を使うのが簡単だ。



zenkaku 拡張機能

この拡張機能をインストールして、コマンドパレットから [Enable Zenkaku] コマンドを実行すると、全角の空白文字が描画されるようになる。また、[Disable Zenkaku] コマンドを実行すると描画されなくなる。

The screenshot shows the Visual Studio Code interface with the title bar "readme.txt - vsctips0021 - Visual Studio Code". The menu bar includes "ファイル(F)", "編集(E)", "選択(S)", "表示(V)", "移動(G)", "デバッグ(D)", "ターミナル(T)", and "ヘルプ(H)". On the left, there's a sidebar with icons for file, search, and settings. The main editor area has a file named "_init_.py" open, containing the following code:

```
1 Enable Zenkaku
2
3     この行は先頭に半角空白文字（ ）が入っています。←これは半角空白文字
4
```

The status bar at the bottom indicates "行 4、列 1 スペース: 2 UTF-8 CRLF プレーンテキスト". There are also icons for file count (6), changes (▲0), and notifications.

全角の空白文字の表示を切り替えているところ

コードやドキュメントを記述していると、空白文字がどこにあるかを知りたいときや、あるいはタブ文字と空白文字（さらには全角空白文字）が混在していると都合が悪いとき、単純に気持ち悪いときなどがある。そうしたときには、`editor.renderWhitespace` 項目や `zenkaku` 拡張機能を使って、空白文字を一目で分かるようにしよう。

VS Code をアンインストールするには

何らかの都合で VS Code アンインストールするときには、VS Code だけをアンインストールする方法と、設定や拡張機能まで削除する方法がある。

Visual Studio Code (以下、VS Code) をアンインストールするには、VS Code 本体のみアンインストールする方法と、VS Code 本体に加えて設定や拡張機能もアンインストールする方法の 2 つの選択肢がある。

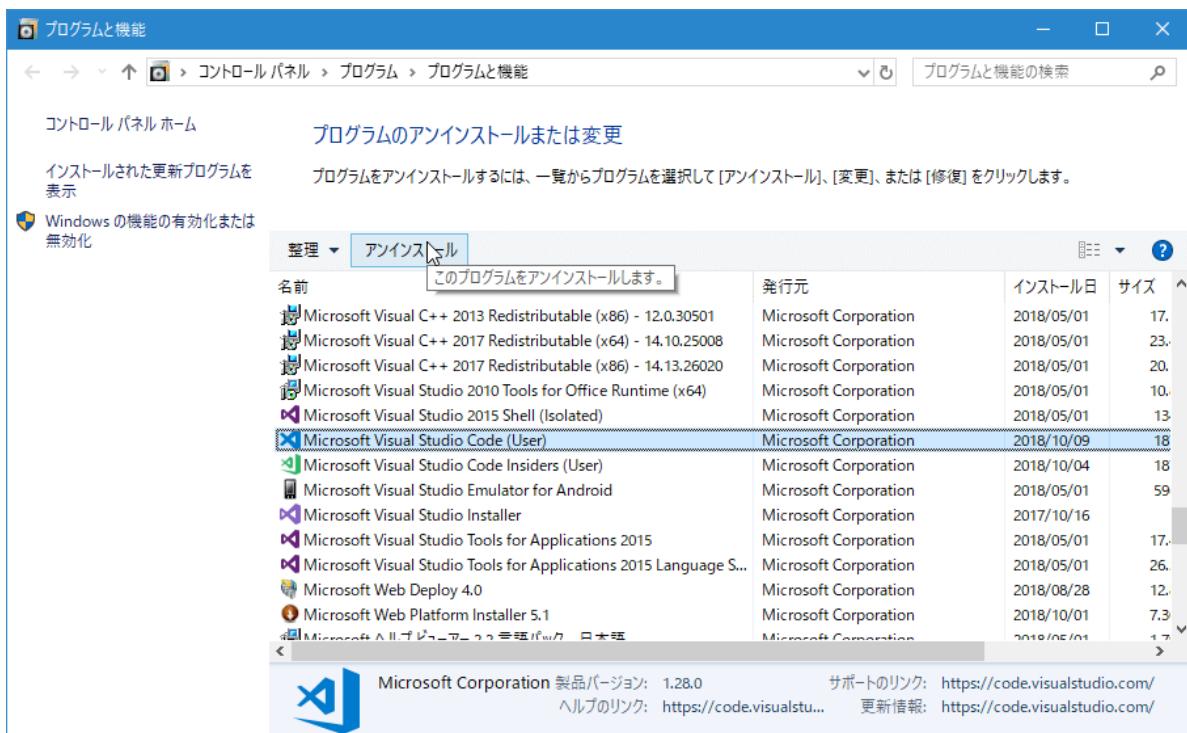
操作	
VS Code 本体をアンインストールする	プラットフォームごとの手順でアプリをアンインストールする
ユーザー設定や拡張機能を削除する	VS Code 本体をアンインストールした後で、設定ファイルを含むフォルダと、拡張機能のインストールフォルダを削除する

VS Code をアンインストールする方法

VS Code 本体をアンインストールする

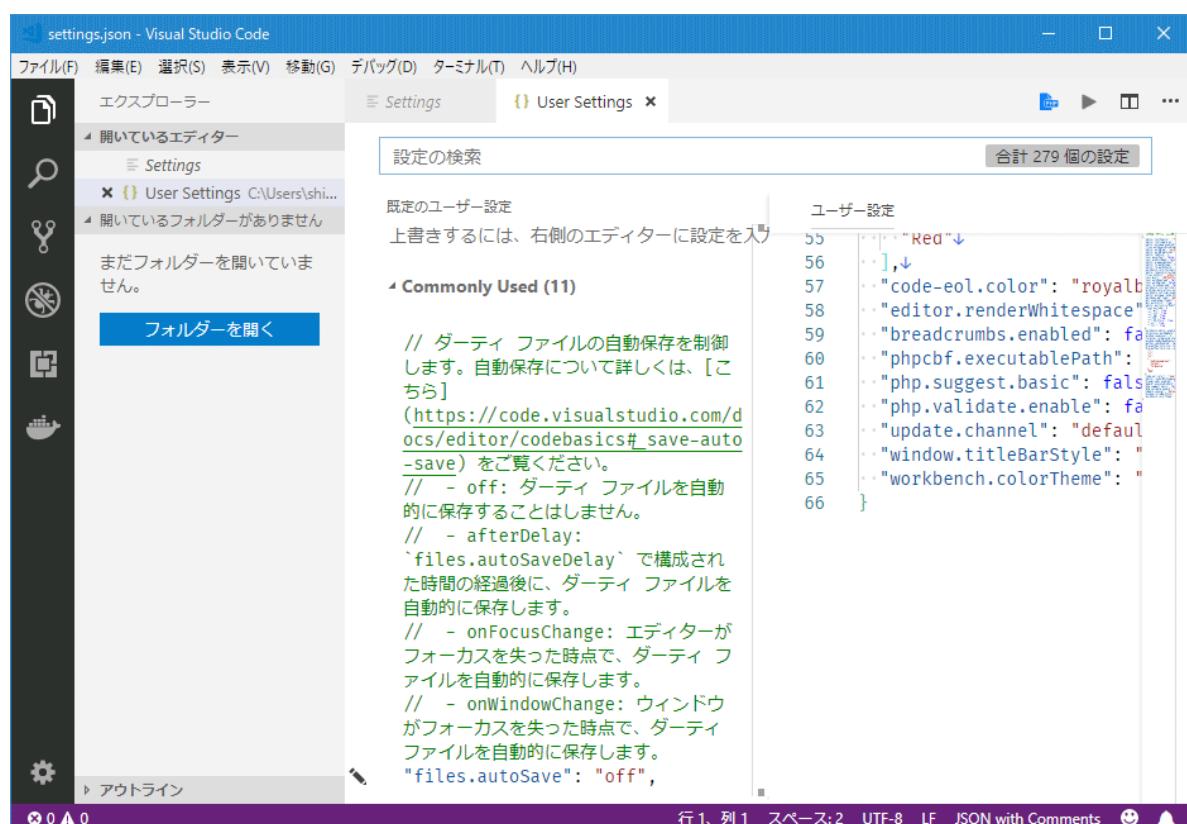
VS Code 本体のアンインストールは、各 OS プラットフォームで一般的なアンインストール手順を実行すればよい。

- Windows : コントロールパネルからアンインストールする
- macOS : [アプリケーション] フォルダからゴミ箱に、VS Code のアイコンをドラッグ&ドロップする
- Linux: 各ディストリビューションで一般的なアンインストール手順を実行する。Ubuntu なら apt コマンドを使う



Windows ではコントロールパネルで、VS Code を選択し、[アンインストール] ボタンをクリックする

上記の手順でアンインストールされるのは、VS Code 本体だけとなる。アンインストール後に VS Code をもう一度インストールすると、以前に行ったユーザー設定や以前にインストールした拡張機能が復活する。



Windows で VS Code をアンインストール後、再インストールしたところ。以前に行った設定や、以前にインストールした拡張機能が復活している

これらも削除したいのであれば、以下の手順をさらに実行する必要がある。

ユーザー設定や拡張機能を削除する

ユーザー設定（`settings.json` ファイル）や表示言語設定（`locale.json` ファイル）などは、OS ごとに異なるフォルダに保存されている。

- Windows : %APPDATA%\Code\User フォルダ
- macOS : ~/Library/Application Support/Code/User フォルダ
- Linux : ~/.config/Code/User フォルダ

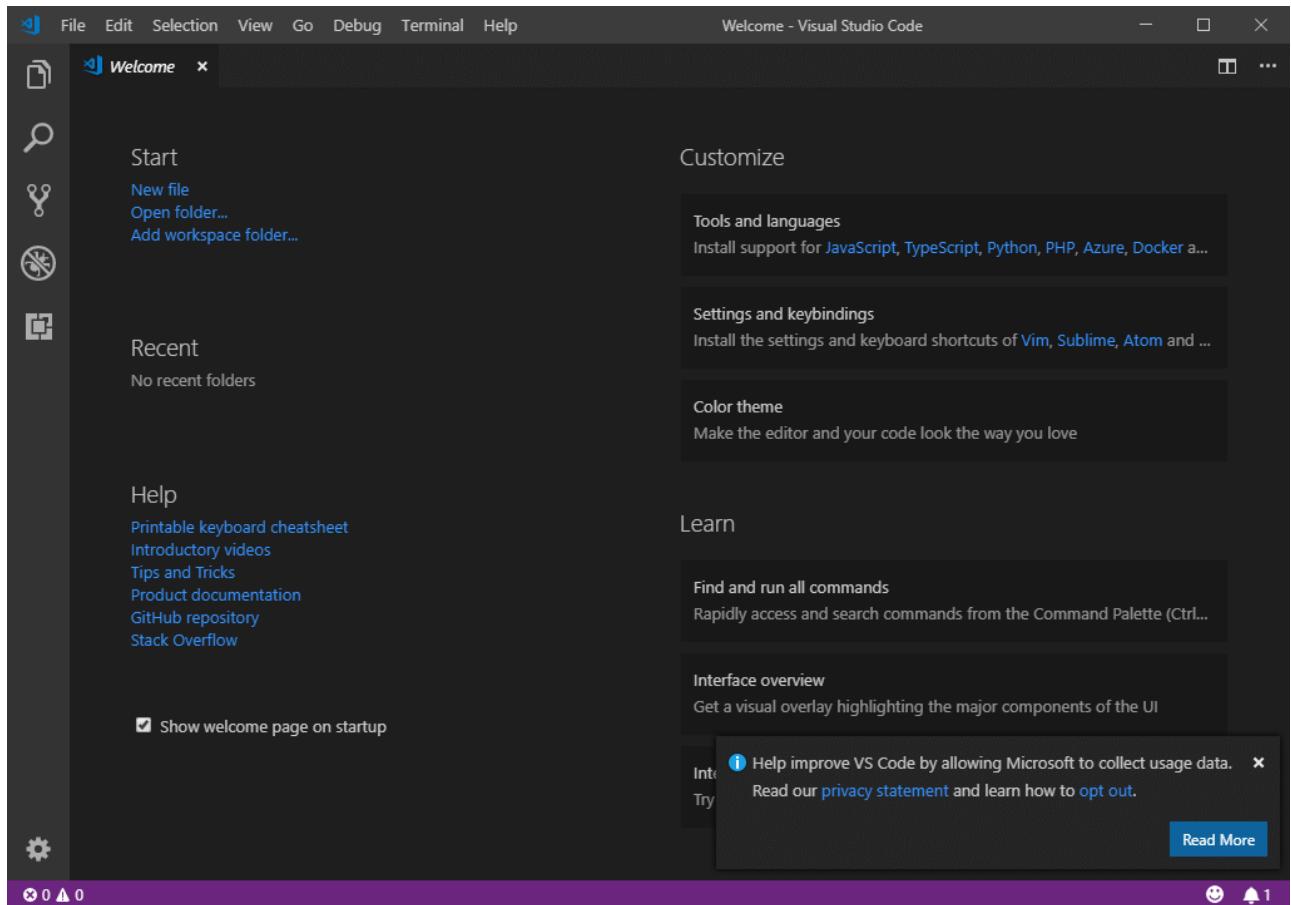
Windows の %APPDATA% 環境変数の値は一般に「C:\Users\< ユーザー名 >\AppData\Roaming」となる。このフォルダにある「Code\User」フォルダにユーザー設定（`settings.json` ファイル）、表示言語設定（`locale.json` ファイル）、ユーザーが設定したキーボードショートカット設定（`keybindings.json` ファイル）などが保存されている。macOS と Linux のユーザーのホームディレクトリを掘っていけば該当するフォルダ／ファイルが存在する。なお、VS Code に関する設定やキャッシュなどは、User フォルダの親フォルダ（Code フォルダ）以下に格納されている（なので、Code フォルダごと削除するのがよいだろう）。また、Insiders ビルドを使用している場合は、「Code」ではなく「Code - Insiders」という名前になる。

同様に、拡張機能のインストールフォルダも OS ごとに異なる。

- Windows : %USERPROFILE%\.vscode\extensions
- macOS : ~/.vscode/extensions
- Linux : ~/.vscode/extensions

上記と同様に、Windows の %USERPROFILE% 環境変数の値は一般に「C:\Users\< ユーザー名 >」となる。つまり、Windows / macOS / Linux のそれぞれでユーザーのホームディレクトリ以下にある「.vscode/extensions」フォルダに拡張機能がインストールされる（Insiders ビルドの場合は「.vscode」ではなく「.vscode-insiders」という名前になる）。

ユーザー設定や拡張機能を削除するには、これらのフォルダを削除する。VS Code をアンインストールして、Code フォルダと `.vscode` フォルダを削除（する代わりにここでは例なのでリネーム）した後で VS Code をインストールすると、全てがデフォルト設定の状態で VS Code が起動するようになる。



全てがデフォルト設定の状態で起動した VS Code (Windows 版)

実際には他にも設定ファイルが残っているかもしれない（例えば、macOS の「~/Library/Preferences/com.microsoft.VSCode*.plist」など）が、各種設定や拡張機能を削除するには上記のフォルダを削除すればよいだろう。

VS Code のミニマップの表示／非表示を切り替えるに

ミニマップは便利だが煩わしいこともある。そんなときには `editor.minimap.enabled` 項目を設定することで表示／非表示を切り替えられる。

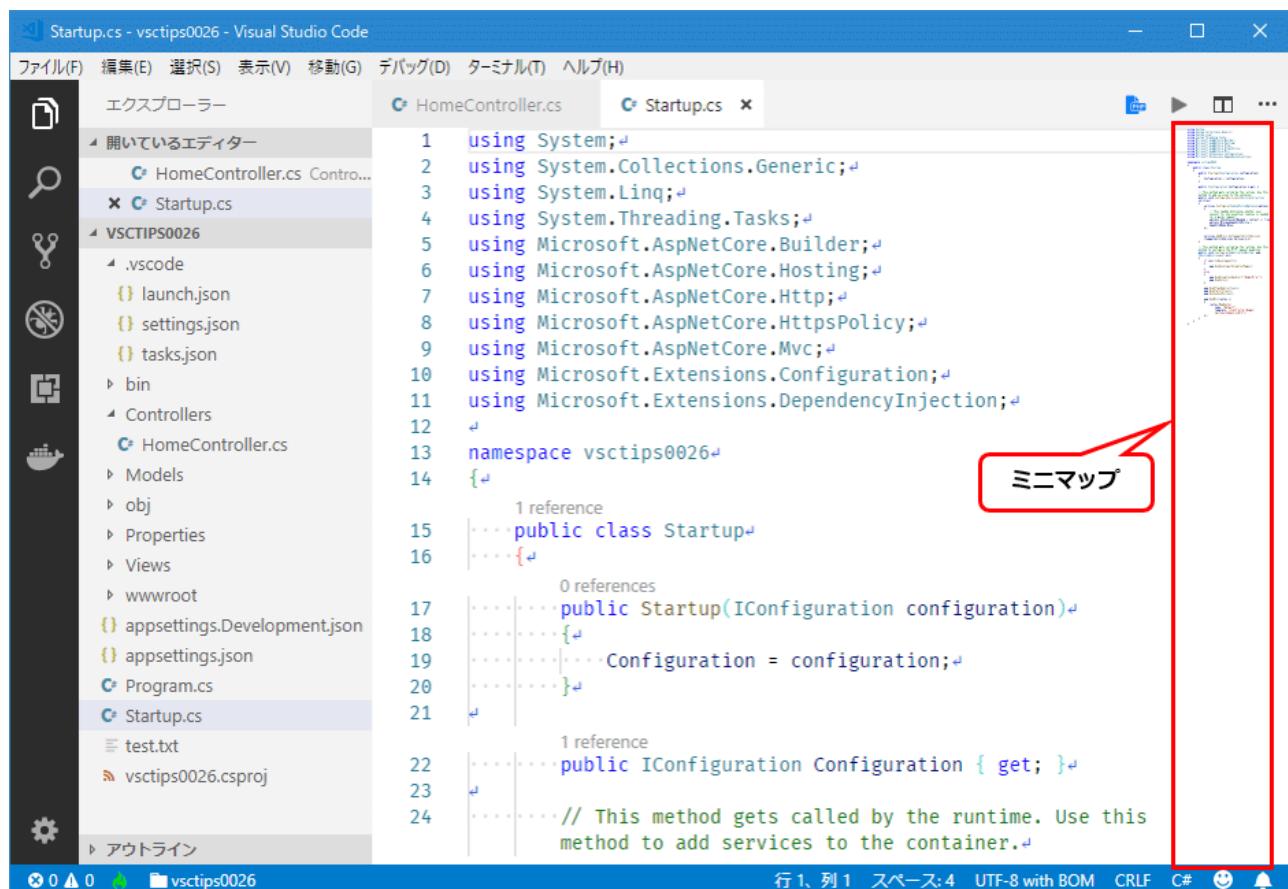
Visual Studio Code (以下、VS Code) には「[ミニマップ](#)」と呼ばれる機能がある。これはデフォルトでスクロールバーの隣に表示される、ドキュメント全体のアウトライン（概要）を示してくれるものだ。だが、これを不要と感じることもある。本稿ではミニマップの表示を制御する方法を示す。

操作
ユーザー設定／ワークスペース設定でミニマップの表示／非表示を切り替える
キーボードからミニマップの表示／非表示を切り替える

ミニマップ関連の設定

ユーザー設定／ワークスペース設定でマップの表示／非表示を切り替える

既に述べたように、ミニマップは現在エディタでアクティブになっているファイルのアウトラインを（デフォルトでは）スクロールバー隣の領域で示してくれるものだ。

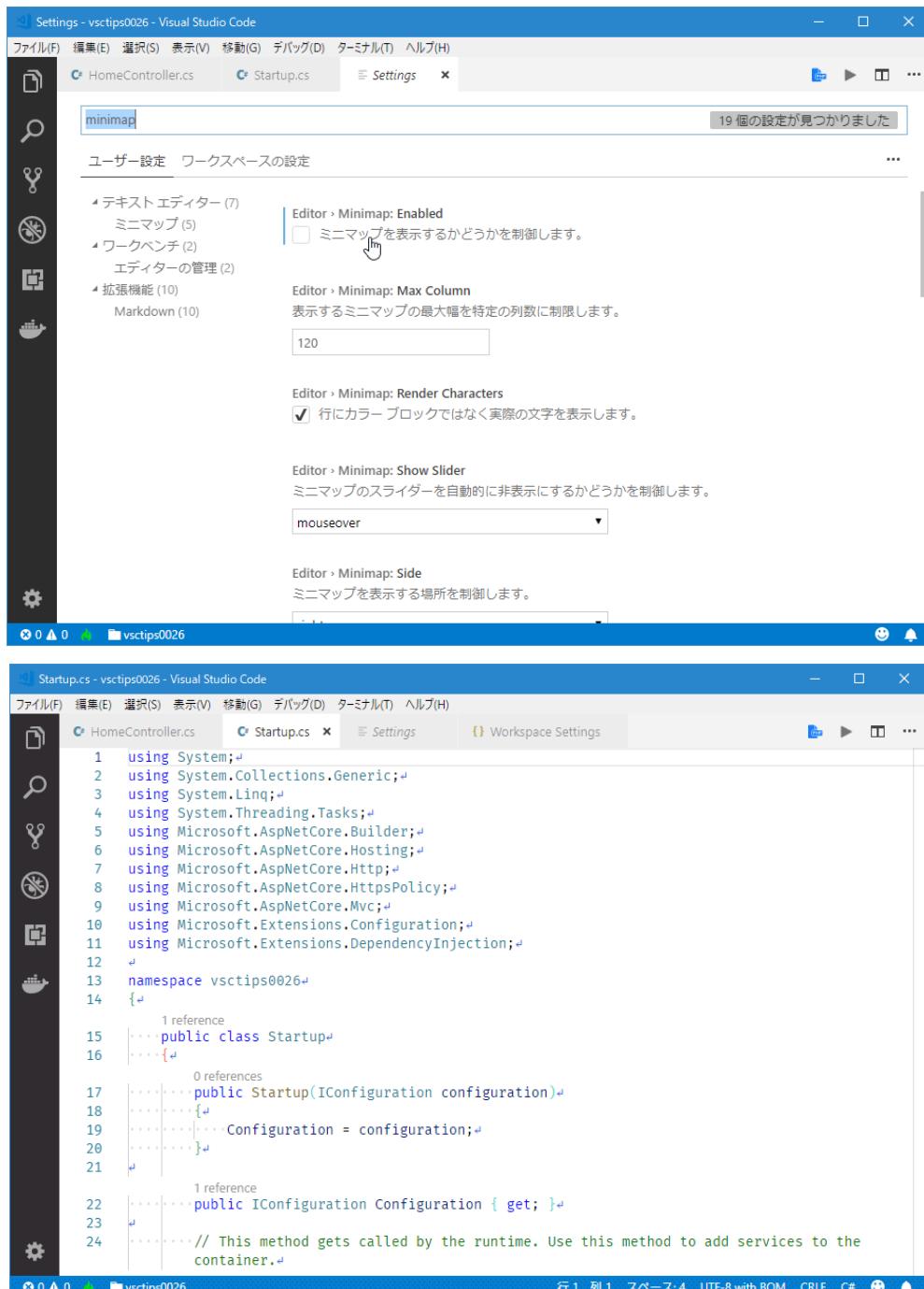


ミニマップがウィンドウ右側に表示されているところ

しかし、「表示領域を少しでも広くしたい」などの理由からこれを表示したくないこともある。そのようなときは、ユーザー設定／ワークスペース設定の `editor.minimap.enabled` 項目を設定することで、表示／非表示を切り替えられる。設定可能な値は次の通り。

- `true` : ミニマップを表示する
- `false` : ミニマップを表示しない

以下に新しい設定エディタでこれを設定している様子を示す。

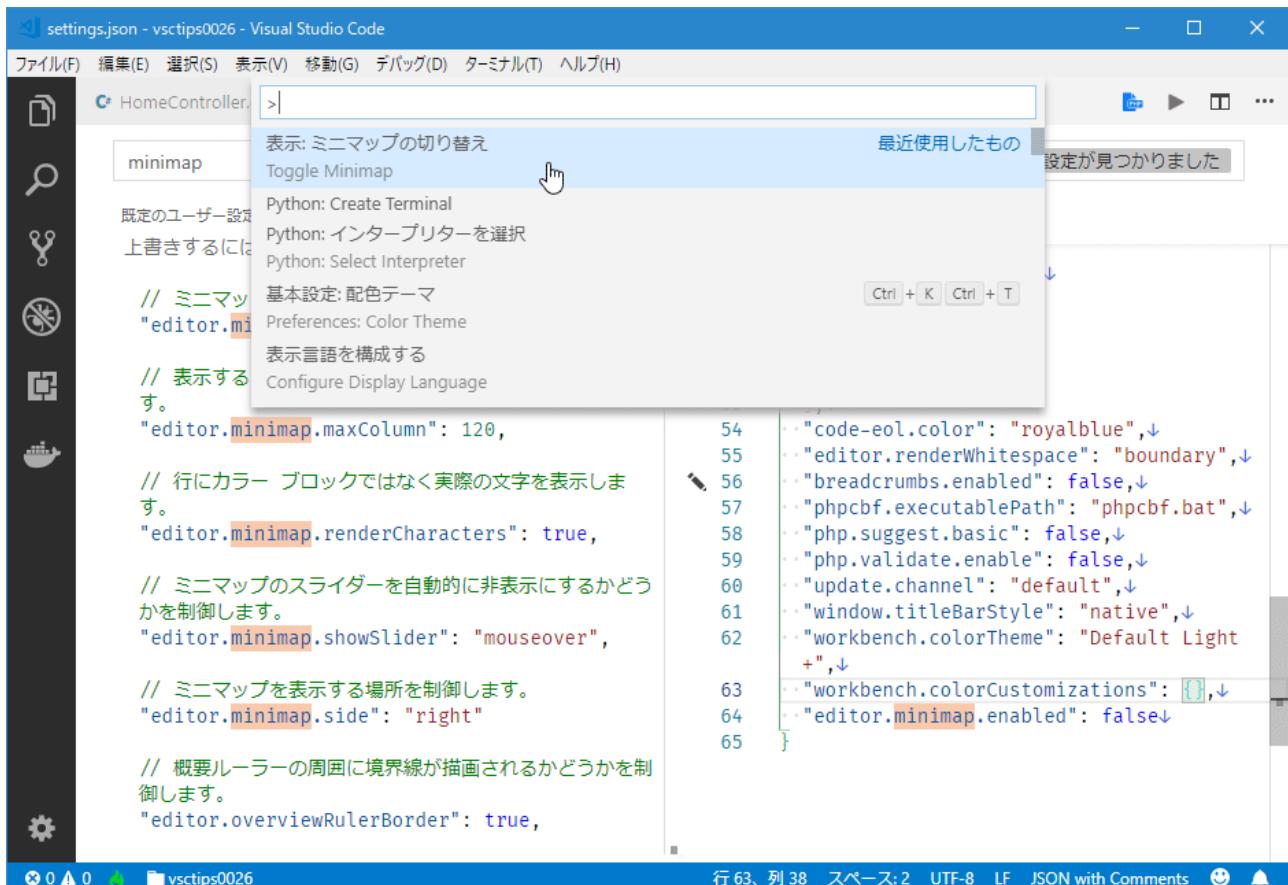


上: 新しい設定エディタでの設定の様子。

下: ミニマップが表示されなくなったところ。

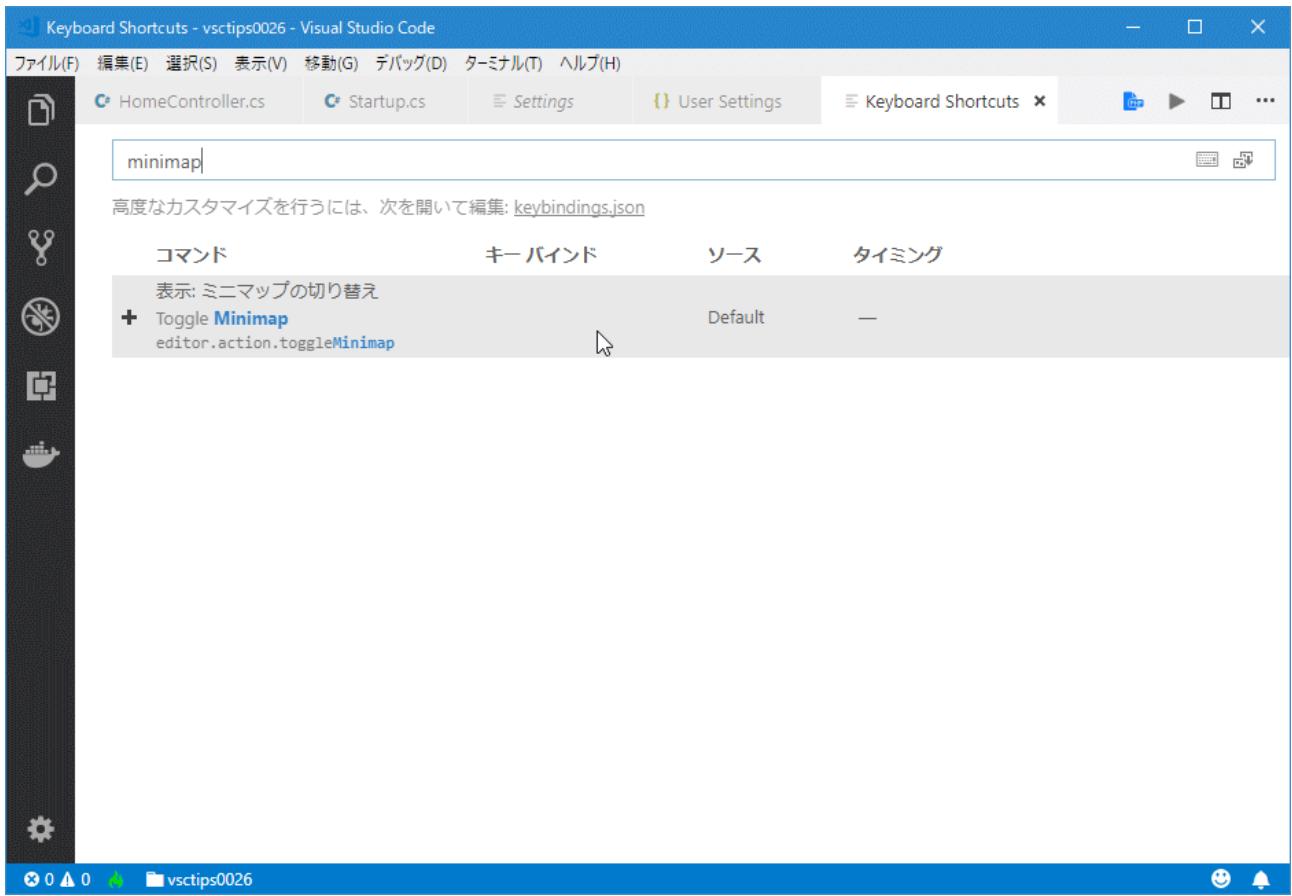
キーボードからミニマップの表示／非表示を切り替える

VS Code のメニュー／コマンドパレットから表示／非表示を切り替えることも可能だ。これには [表示] メニューの [ミニマップの切り替え] を選択するか、コマンドパレットで「minimap」などと入力して [表示：ミニマップの切り替え] コマンドを選択すればよい。



コマンドパレットでミニマップの表示／非表示を切り替えているところ

これらのメニュー項目／コマンドは実際には、上に示した `editor.minimap.enabled` 項目を設定するためのショートカットとなっている。キーボードショートカットを割り当てることも可能なので、ミニマップの表示／非表示を頻繁に切り替えるようなら、設定をしておこう。



ミニマップの表示／非表示の切り替えにはキーボードショートカットを割り当て可能

注意したいのは、これらのメニュー項目／コマンドは「ユーザー設定」レベルで `editor.minimap.enabled` 項目を設定するものである点だ。ワークスペース設定（現在、VS Code で開いているフォルダにある `.vscode` フォルダ内の `settings.json` ファイル）はユーザー設定よりも優先されるため、既にこの項目をワークスペースで設定している場合には、キーボードからこれらを選択してもミニマップの表示／非表示が切り替わらない。

ユーザー設定／ワークスペース設定では、この他にもミニマップの表示位置を指定する `editor.minimap.side` 項目などがある。興味のある方は、ユーザー設定／ワークスペース設定で「minimap」を検索して、各項目の説明を読みながら、実際に試してみよう。

VS Code のミニマップの表示方法を変更するには

VS Code のミニマップはその表示／非表示を切り替える以外にも、さまざまな形でカスタマイズできる。その方法と、実際の表示の違いを説明する。

Visual Studio Code (以下、VS Code) のミニマップは `editor.minimap.enabled` 項目の設定により、表示／非表示を切り替えられるが、その他にも VS Code のウィンドウ内での表示位置を変更したり、ミニマップに文字／ブロックのどちらを表示したりするなどを指定できる。本稿ではこれらの項目について説明していこう。

	操作
ミニマップの表示位置を切り替える	<code>editor.minimap.side</code> 項目を設定する "right" : ウィンドウ右側に表示（デフォルト） "left" : ウィンドウ左側に表示
ミニマップに文字／ブロックのどちらを表示するかを指定する	<code>editor.minimap.renderCharacters</code> 項目を設定する true : 文字を表示（デフォルト） ブロックを表示
ミニマップの最大幅（文字数）を指定する	<code>editor.minimap.maxColumn</code> 項目を設定する デフォルト値は120
ミニマップのスライダーの表示方法を指定する	<code>editor.minimap.showSlider</code> 項目を設定する "mouseover" : マウスカーソルがミニマップ上にあるときに表示（デフォルト） "always" : 常に表示

ミニマップの表示方法を変更する方法

ミニマップの表示位置を切り替える

ミニマップはウィンドウの右側と左側のどちらに表示するかを設定できる。これはユーザー設定／ワークスペース設定の `editor.minimap.side` 項目で指定する。指定可能な値は次の通りだ。

- "right" : 右側に表示（デフォルト）
- "left" : 左側に表示

この項目により表示がどう変わるかを以下に示す。

Startup.cs - vsctips0027 - Visual Studio Code

ファイル(F) 編集(E) 選択(S) 表示(V) 移動(G) デバッグ(D) ターミナル(T) ヘルプ(H)

エクスプローラー

開いているエディター

- Startup.cs
- VSCTIPS0027
- .vscode
- bin
- Controllers
- Models
- obj
- Properties
- Views
- wwwroot
- appsettings.Development.json
- appsettings.json
- Program.cs
- Startup.cs
- vsctips0027.csproj

Startup.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Threading.Tasks;
5  using Microsoft.AspNetCore.Builder;
6  using Microsoft.AspNetCore.Hosting;
7  using Microsoft.AspNetCore.Http;
8  using Microsoft.AspNetCore.HttpsPolicy;
9  using Microsoft.AspNetCore.Mvc;
10 using Microsoft.Extensions.Configuration;
11 using Microsoft.Extensions.DependencyInjection;
12 
13 namespace vsctips0027
14 {
15     public class Startup
16     {
17         public Startup(IConfiguration configuration)
18         {
19             Configuration = configuration;
20         }
21     }
22     public IConfiguration Configuration { get; }
23     // This method gets called by the runtime. Use this method to add services to the container.
24 }
```

行 12、列 1 スペース:4 UTF-8 with BOM CRLF C# 😊 🔔

Startup.cs - vsctips0027 - Visual Studio Code

ファイル(F) 編集(E) 選択(S) 表示(V) 移動(G) デバッグ(D) ターミナル(T) ヘルプ(H)

エクスプローラー

開いているエディター

- Startup.cs
- Workspace Settings .vscode
- VSCTIPS0027
- .vscode
- launch.json
- settings.json
- tasks.json
- bin
- Controllers
- Models
- obj
- Properties
- Views
- wwwroot
- appsettings.Development.json
- appsettings.json
- Program.cs
- Startup.cs
- vsctips0027.csproj

Workspace Settings

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Threading.Tasks;
5  using Microsoft.AspNetCore.Builder;
6  using Microsoft.AspNetCore.Hosting;
7  using Microsoft.AspNetCore.Http;
8  using Microsoft.AspNetCore.HttpsPolicy;
9  using Microsoft.AspNetCore.Mvc;
10 using Microsoft.Extensions.Configuration;
11 using Microsoft.Extensions.DependencyInjection;
12 
13 namespace vsctips0027
14 {
15     public class Startup
16     {
17         public Startup(IConfiguration configuration)
18         {
19             Configuration = configuration;
20         }
21     }
22     public IConfiguration Configuration { get; }
23     // This method gets called by the runtime. Use this method to add services to the container.
24 }
```

行 12、列 1 スペース:4 UTF-8 with BOM CRLF C# 😊 🔔

上 : editor.minimap.side 項目を "right" にした場合 (デフォルト)。

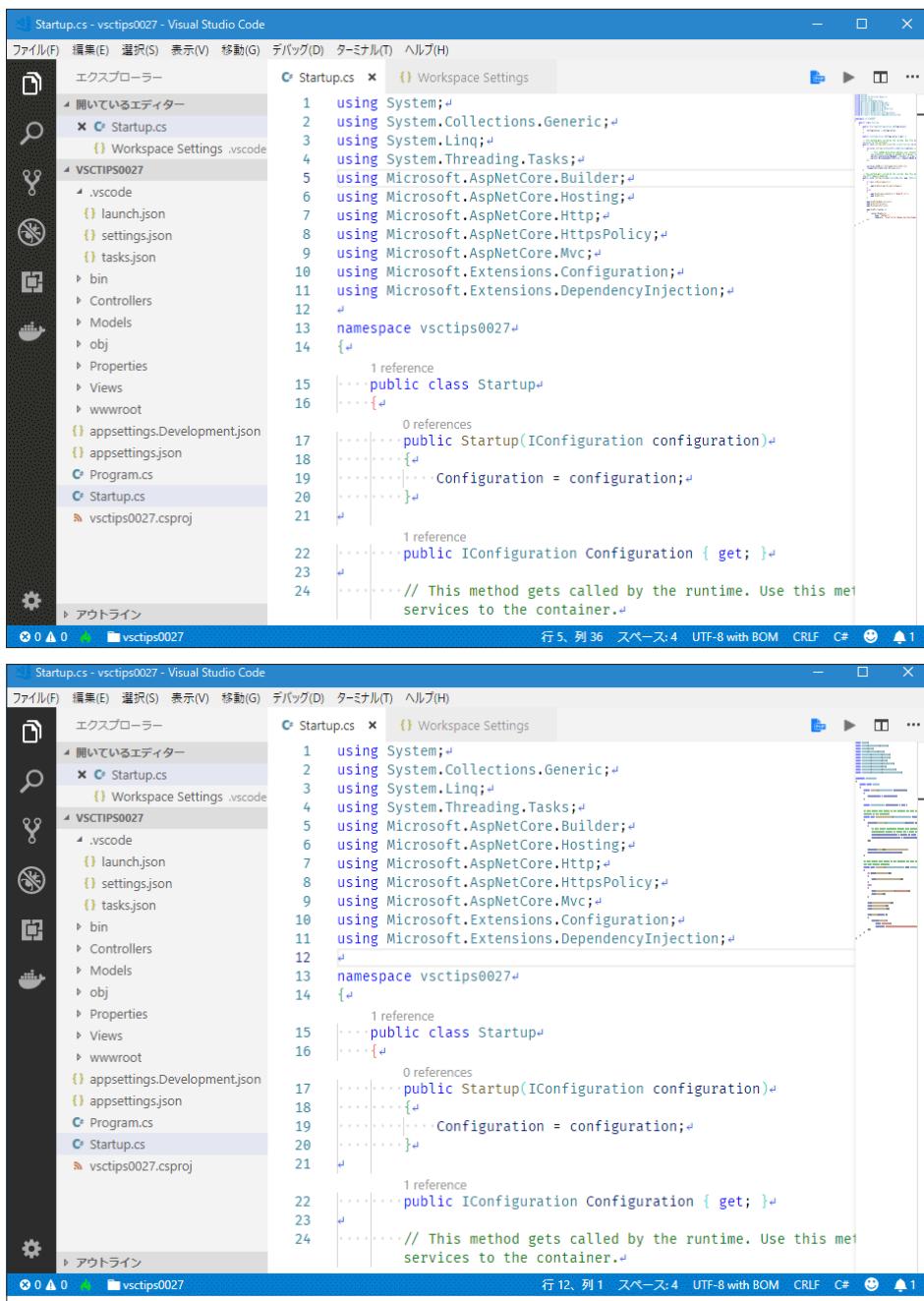
下 : editor.minimap.side 項目を "left" にした場合。

ミニマップに文字／ブロックのどちらを表示するかを指定する

ミニマップにエディタに開かれているファイルの概要が表示されるが、その際に実際の文字（を縮小したイメージ）として表示するか、色付きのブロックとして表示するかを指定できる。これはユーザー設定／ワークスペース設定の `editor.minimap.renderCharacters` 項目で指定する。指定可能な値は次の通り。

- `true` : ミニマップに文字を表示
- `false` : ミニマップにブロックを表示

この項目により表示がどう変わるかを以下に示す。



上 : `editor.minimap.renderCharacters` 項目を `true` にした場合（デフォルト）。

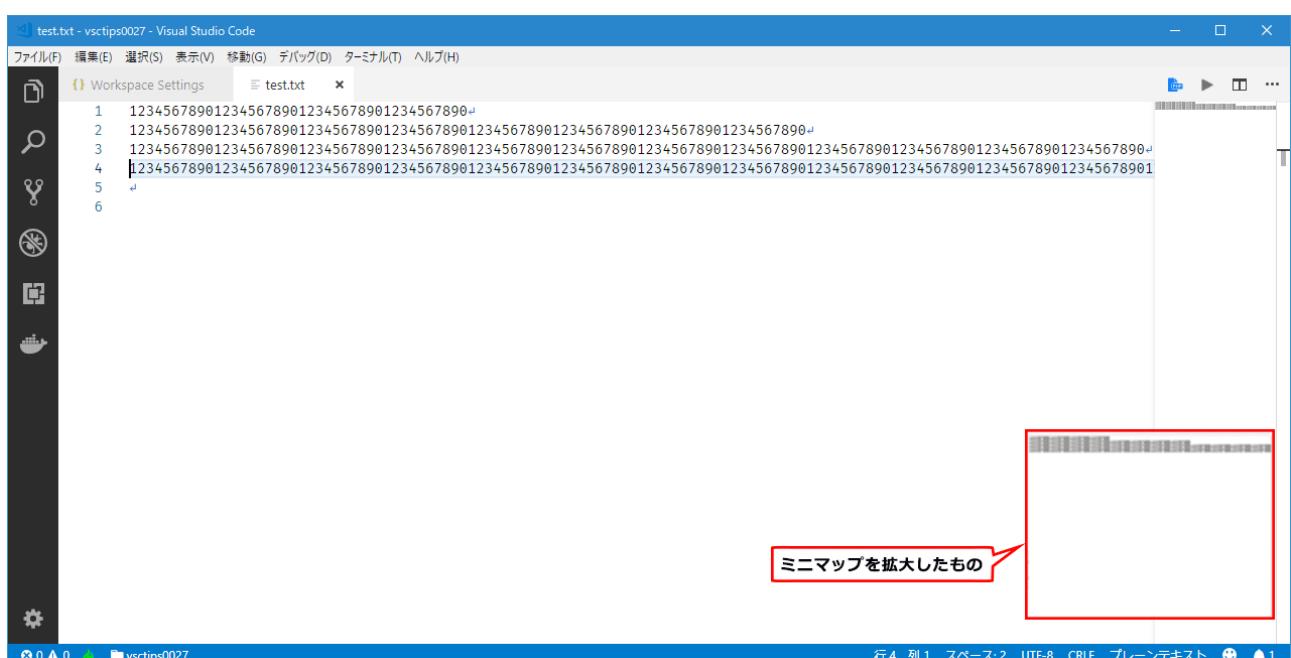
下 : `editor.minimap.renderCharacters` 項目を `false` にした場合。

文字を表示した方が実際のコードとミニマップ表示のイメージは何となく近いものになるが、ブロックを表示した場合にはミニマップに示されるコード全体の概要是濃淡が濃くなり、よりハッキリとしたものになる。どちらにするかは好みといえる。

ミニマップの最大幅（文字数）を指定する

ミニマップに表示する概要の最大幅（文字数）は `editor.minimap.maxColumn` 項目で指定できる。デフォルト値は「120」（文字）となっている。

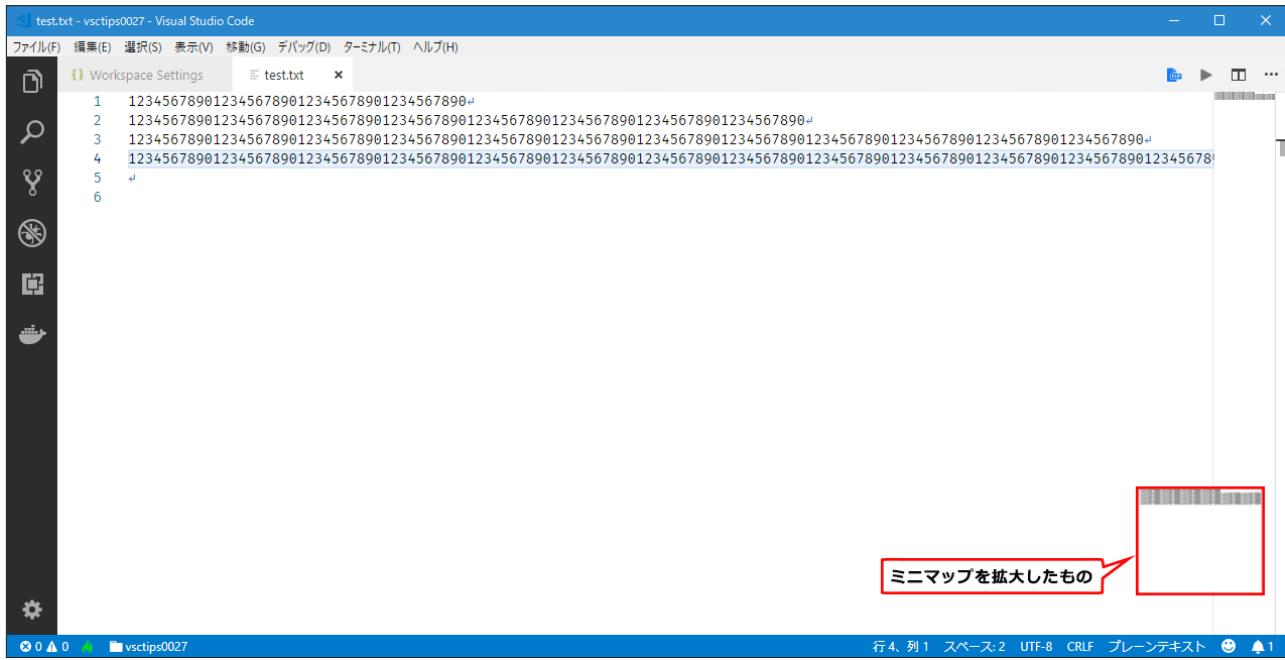
以下は `editor.wordWrap` 項目を "off" にした状態で、1 行目が 40 文字、2 行目が 80 文字、3 行目が 120 文字、4 行目が 160 文字あるテキストファイルをエディタに表示したところである。`editor.minimap.maxColumn` 項目の値は設定していない（デフォルト値の「120」が有効）。



デフォルトではミニマップには 120 文字まで文字（あるいはブロック）が表示される

ミニマップ（あるいは、画像右下にあるミニマップを拡大したもの）をよく見てもらうと分かるが、最初の行は全体の 3 分の 1 の幅（120 文字中の 40 文字）でイメージが表示されている。2 行目も同様に全体の 3 分の 2 の幅、つまり 80 文字のイメージが表示されている。3 行目は 120 文字なので幅一杯に表示が行われている。4 行目は 160 文字あるが、121 文字目以降は表示されなくなる。

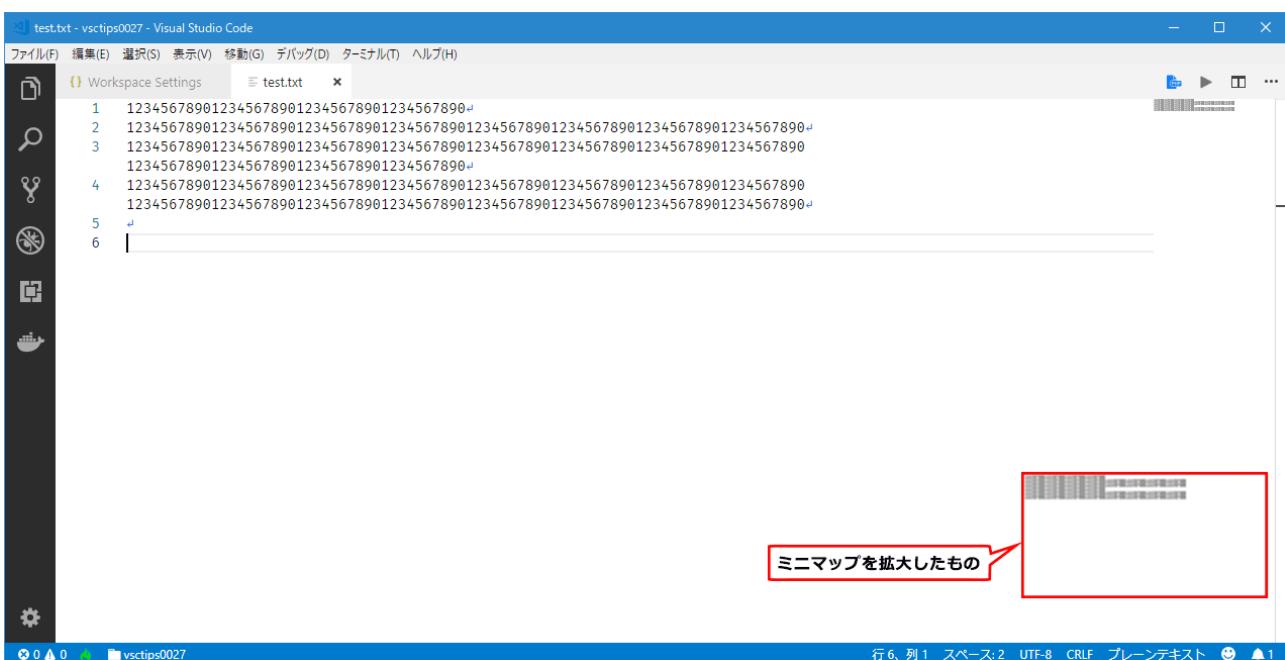
`editor.minimap.maxColumn` 項目の値を「60」にしてみると、次のようになる。



editor.minimap.maxColumn 項目を「60」にした場合

これもミニマップまたは画像右下のミニマップを拡大したものをよく見てみよう。まず、ミニマップ自体の幅が狭くなっている。また、ミニマップ上で 1 行目が全体の 3 分の 2 の幅（60 文字中の 40 文字）まで表示され、2 行目以降は 61 文字目以降が表示されなくなる。

なお、ミニマップ表示は `editor.wordWrap` 項目の影響も受けることには注意しよう。つまり、エディタで折り返しが有効になっていれば、ミニマップでも折り返しが反映される。例えば、`editor.wordWrap` 項目の値を "wordWrapColumn" に、`editor.wordWrapColumn` 項目の値を「80」にすると、「80 文字で折り返し」が行われるようになる（筆者の通常の設定）。この場合には、以下のような表示になる。

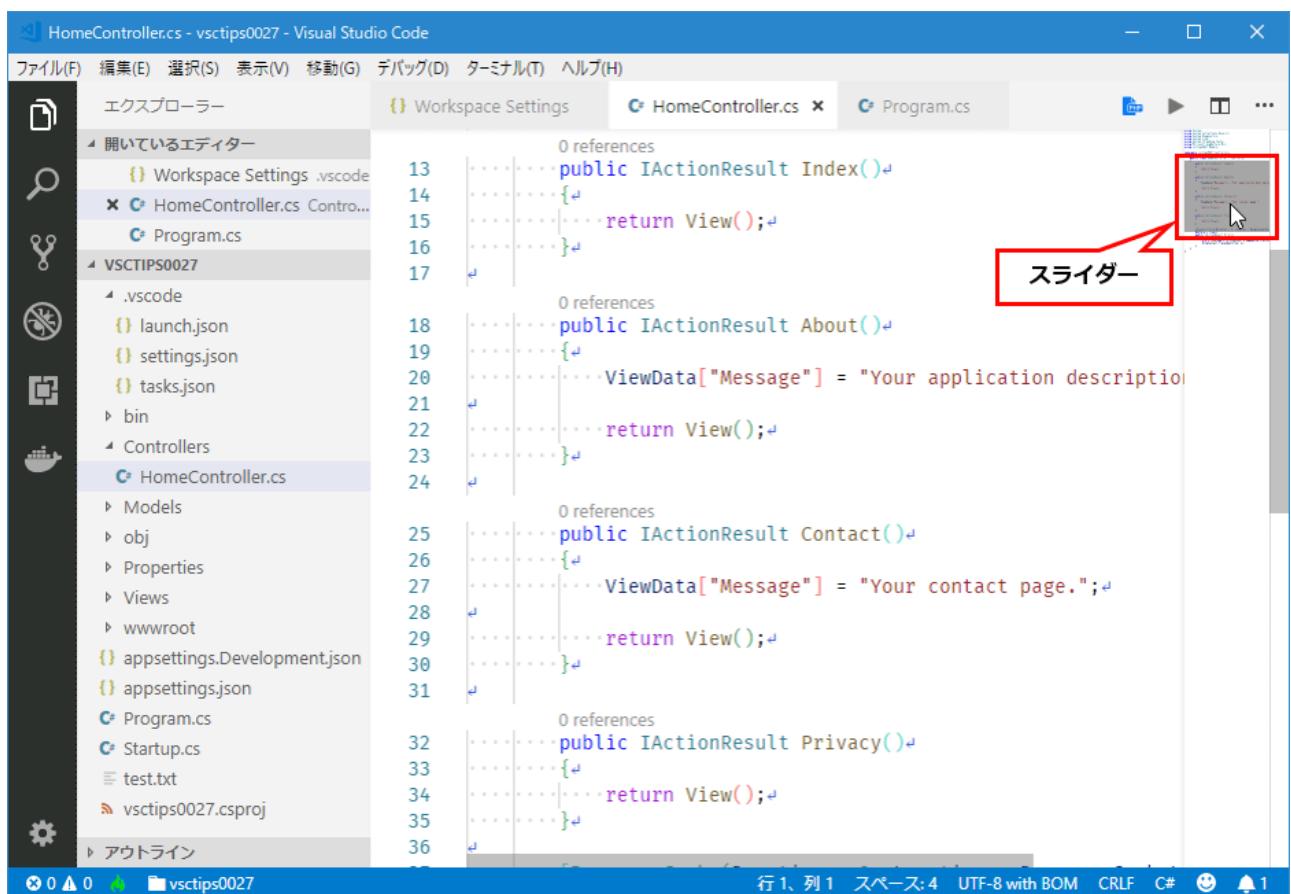


ミニマップの表示はTディタでの折り返しを反映する

ミニマップ（または画像右下のミニマップを拡大したもの）を見ると、右端には余白が全体の3分の1程度ある。これは editor.minimap.maxColumn 項目の大さきが「120」であるのに対して、エディタでは「80」文字で折り返しが行われるため、結果としてミニマップの3分の2（120文字中の80文字）が概要の表示に使われているということだ。

ミニマップのスライダーの表示方法を指定する

ミニマップには「スライダー」と呼ばれる領域も表示される。これは、エディタ領域に現在表示されている部分がドキュメント全体のどの辺かを示すものだ。また、スライダーをドラッグすることで、エディタをスクロールさせることも可能だ。



スライダーはエディタ領域に現在表示されている部分がどこかをミニマップ上に示す

スライダーの表示方法は editor.minimap.showSlider 項目で以下のどちらかを選択することで指定できる。

- "mouseover" : ミニマップにマウスカーソルがあるときに表示（デフォルト）
- "always" : スライダーを常に表示

これらの設定の違いでミニマップの表示がどうなるかを以下に示す。

HomeController.cs - vsctips0027 - Visual Studio Code

ファイル(F) 編集(E) 選択(S) 表示(V) 移動(G) デバッグ(D) ターミナル(T) ヘルプ(H)

エクスプローラー Workspace Settings C# HomeController.cs x Program.cs

開いているエディター

- Workspace Settings .vscode
- HomeController.cs Controller
- Program.cs

VSCTIPS0027

- .vscode
- launch.json
- settings.json
- tasks.json
- bin
- Controllers
- HomeController.cs
- Models
- obj
- Properties
- Views
- wwwroot
- appsettings.Development.json
- appsettings.json
- Program.cs
- Startup.cs
- test.txt
- vsctips0027.csproj

アウトライン

行1、列1 スペース:4 UTF-8 with BOM CRLF C# 😊 🔔

```
10 {  
11     0 references  
12     public class HomeController : Controller{  
13         0 references  
14         {  
15             return View();  
16         }  
17     }  
18         0 references  
19         {  
20             ViewData["Message"] = "Your application description.  
21         "  
22             return View();  
23         }  
24     }  
25         0 references  
26         {  
27             ViewData["Message"] = "Your contact page.";  
28         "  
29             return View();  
30         }  
31     }  
32         0 references  
33         public IActionResult Privacy()  
34     }
```

HomeController.cs - vsctips0027 - Visual Studio Code

ファイル(F) 編集(E) 選択(S) 表示(V) 移動(G) デバッグ(D) ターミナル(T) ヘルプ(H)

エクスプローラー Workspace Settings .vscode C# HomeController.cs x Program.cs

開いているエディター

- Workspace Settings .vscode
- HomeController.cs Controller
- Program.cs

VSCTIPS0027

- .vscode
- launch.json
- settings.json
- tasks.json
- bin
- Controllers
- HomeController.cs
- Models
- obj
- Properties
- Views
- wwwroot
- appsettings.Development.json
- appsettings.json
- Program.cs
- Startup.cs
- test.txt
- vsctips0027.csproj

アウトライン

行1、列1 スペース:4 UTF-8 with BOM CRLF C# 😊 🔔

```
10 {  
11     0 references  
12     public class HomeController : Controller{  
13         0 references  
14         {  
15             return View();  
16         }  
17     }  
18         0 references  
19         {  
20             ViewData["Message"] = "Your application description.  
21         "  
22             return View();  
23         }  
24     }  
25         0 references  
26         {  
27             ViewData["Message"] = "Your contact page.";  
28         "  
29             return View();  
30         }  
31     }  
32         0 references  
33         public IActionResult Privacy()  
34     }
```

上: editor.minimap.showSlider 項目の値を "mouseover" にした場合（デフォルト）。

下: editor.minimap.showSlider 項目の値を "always" にした場合。

1つ目の画像は、この項目の値を "mouseover" にしたもの。ミニマップ上にマウスカーソルがないのでスライダーが表示されていない。これに対して2つ目の画像は、この項目を "always" にしたもの。マウスカーソルがミニマップ上にあってもスライダーが表示されている点に注目しよう。

どちらにするかはユーザーの好みだが、サイズが大きなファイルを編集するなら、"always"にしておくと、ファイルのどの辺を現在編集しているかをすぐに把握できるので、こちらに変更してもよいだろう。

VS Code でフォントを変更するには

VS Code で、エディタおよび統合ターミナルに表示するテキストのフォントやそのサイズなどを指定する方法を説明する。

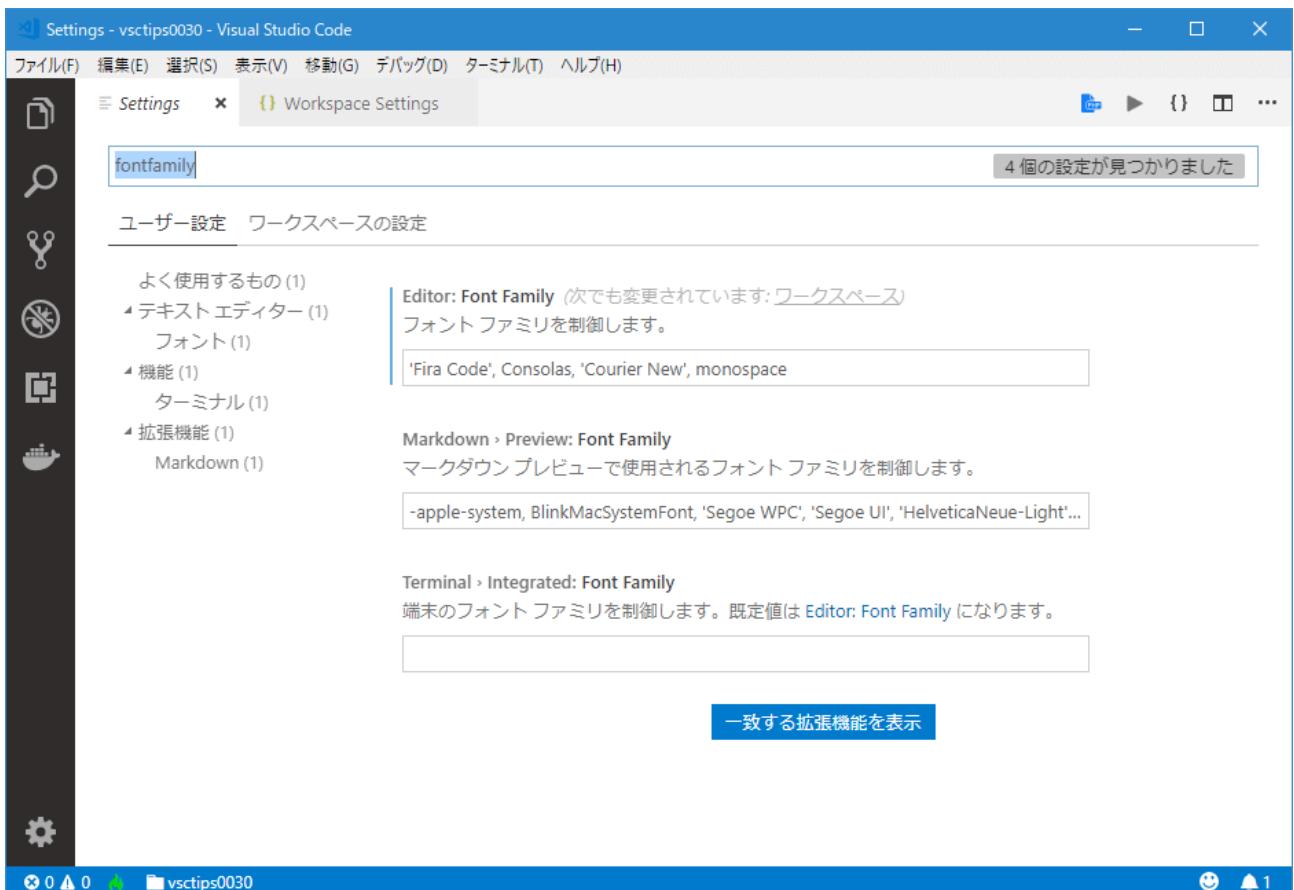
Visual Studio Code (以下、VS Code) では、エディタおよび統合ターミナルで使用するフォントを好みのものを指定できる。

	操作
エディタのフォントを指定する	editor.fontFamily項目で指定する
エディタのフォントサイズを指定する	editor.fontSize項目で指定する
統合ターミナルのフォントを指定する	terminal.integrated.fontFamily項目で指定する
統合ターミナルのフォントサイズを指定する	terminal.integrated.fontSize項目で指定する
その他	フォントの太さを変更する： editor.fontWeight項目／terminal.integrated.fontWeight項目 フォントの合字機能を有効化する： editor.fontLigatures項目（統合ターミナル用の設定項目はなし）

VS Code のフォント関連設定

エディタのフォントを指定する

VS Code でエディタのテキスト表示に使用するフォントを変更するにはユーザー設定／ワークスペース設定で `editor.fontFamily` 項目を設定する。新しい設定エディタで設定する場合も、`settings.json` ファイルで設定する場合も検索ボックスに「`fontfamily`」などと入力するとよい。

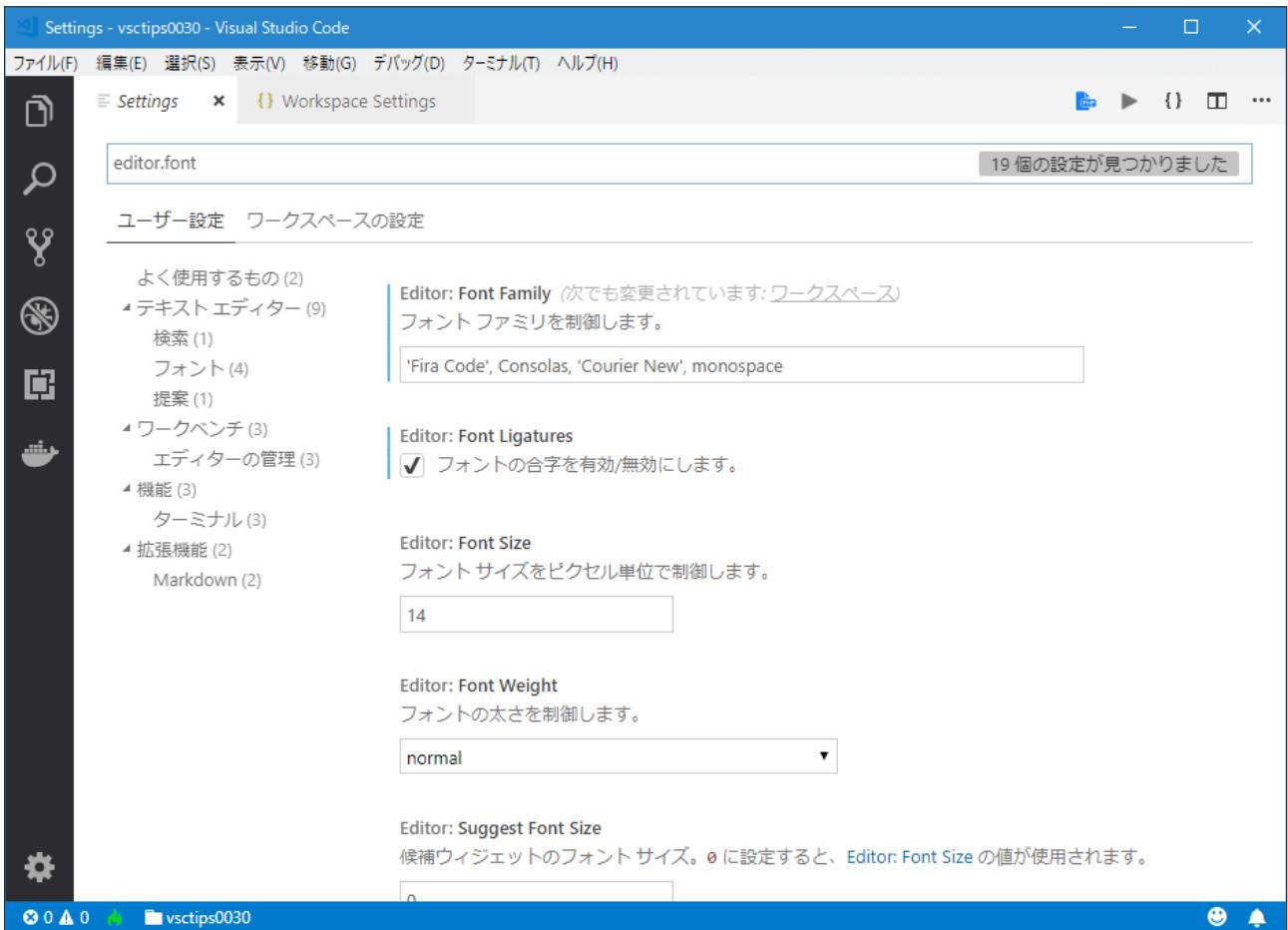


上は新しい設定エディタでユーザー設定を参照しているところ

フォントは複数指定する場合には、カンマで区切る。また、フォント名に空白文字を含む場合、それらは「シングルクオート」で囲む必要がある（例：'Fira Code'）。

なお、フォントには VS Code をインストールしているコンピュータで使用できるものを指定する必要がある。例えば、上の画像で指定している「Fira Code」というフォントは、筆者が独自にインストールしたものだ。

フォントの太さを指定する `editor.fontSize` 項目、フォントの合字機能（特定の並びの文字を合成して 1 つの文字のように表示する機能）の有効／無効を指定する `editor.fontLigatures` 項目もあるので、必要に応じてこれらも設定しておこう。



フォント関連の設定項目を新しい設定エディタで一覧したところ

特に、開発者向けのフォントである Fira Code を指定した場合には、`editor.fontLigatures` 項目を `true` に設定しておくことで、コードが格段に見やすくなる。以下に例を示す。

```
.... static void Main(string[] args) { ... } // == >= <= !=
```

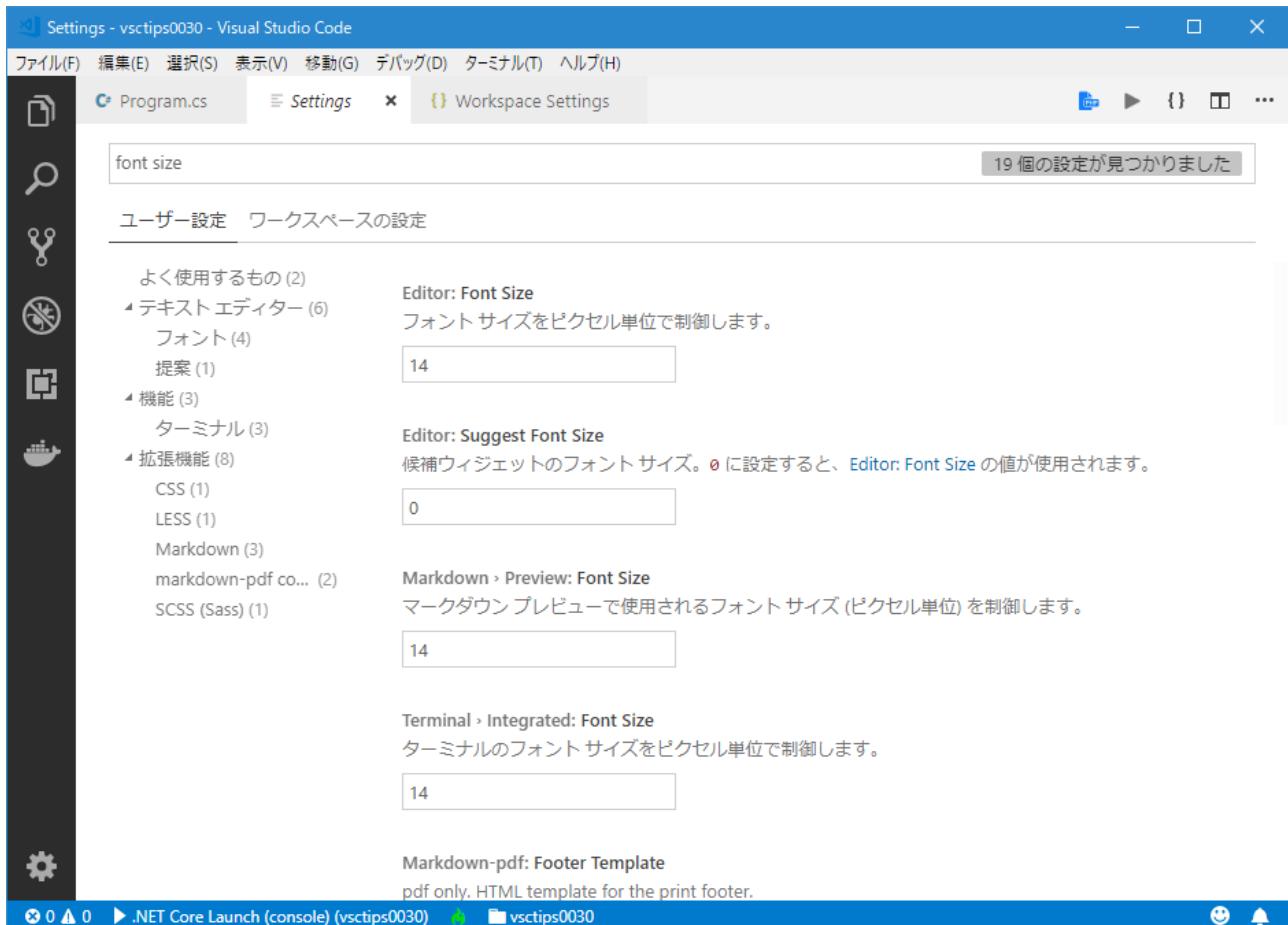
false にした場合。比較演算子が入力したまま表示されている。

```
.... static void Main(string[] args) { ... } // = ≥ ≤ ≠
```

true にした場合。比較演算子が合字機能により見やすい表示になっている。

エディタのフォントサイズを指定する

エディタに表示するテキストのフォントサイズは、ユーザー設定／ワークスペース設定で `editor.fontSize` 項目を指定できる（ピクセル単位）。フォントサイズのデフォルト値は OS ごとに異なる。

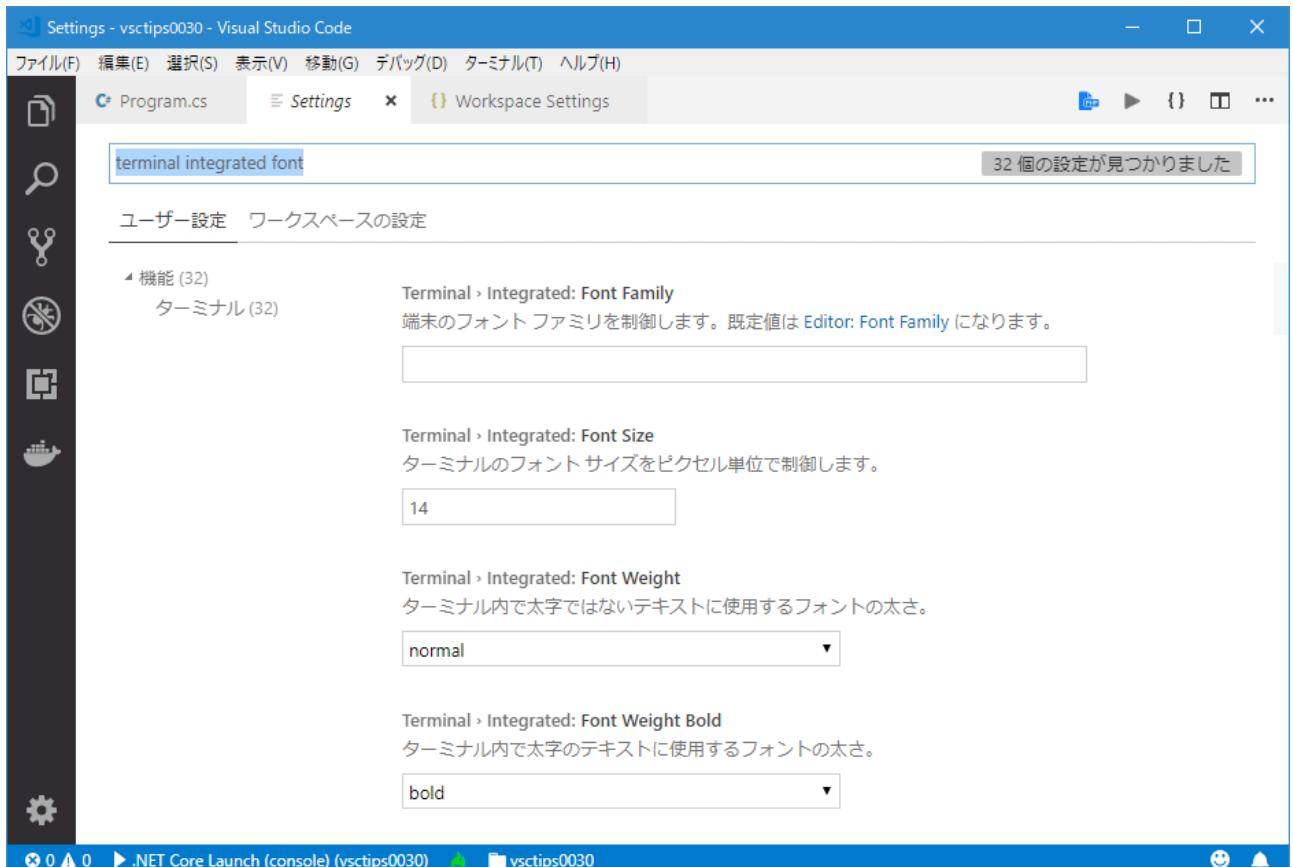


エディタに表示するテキストのフォントサイズ指定（[Editor: Font Size] 項目）

上の画像を見ると分かる通り、IntelliSense が表示する候補のフォントサイズを指定する `editor.fontSize` 項目もある（上から 2 つ目の項目）。デフォルト値は「0」くなっているが、これは `editor.fontSize` 項目の値に従うこと意味する。

統合ターミナルのフォントを指定する

統合ターミナルで使用するテキストのフォントも指定可能だ。これには、ユーザー設定／ワークスペース設定で `terminal.integrated.fontFamily` 項目を設定する。



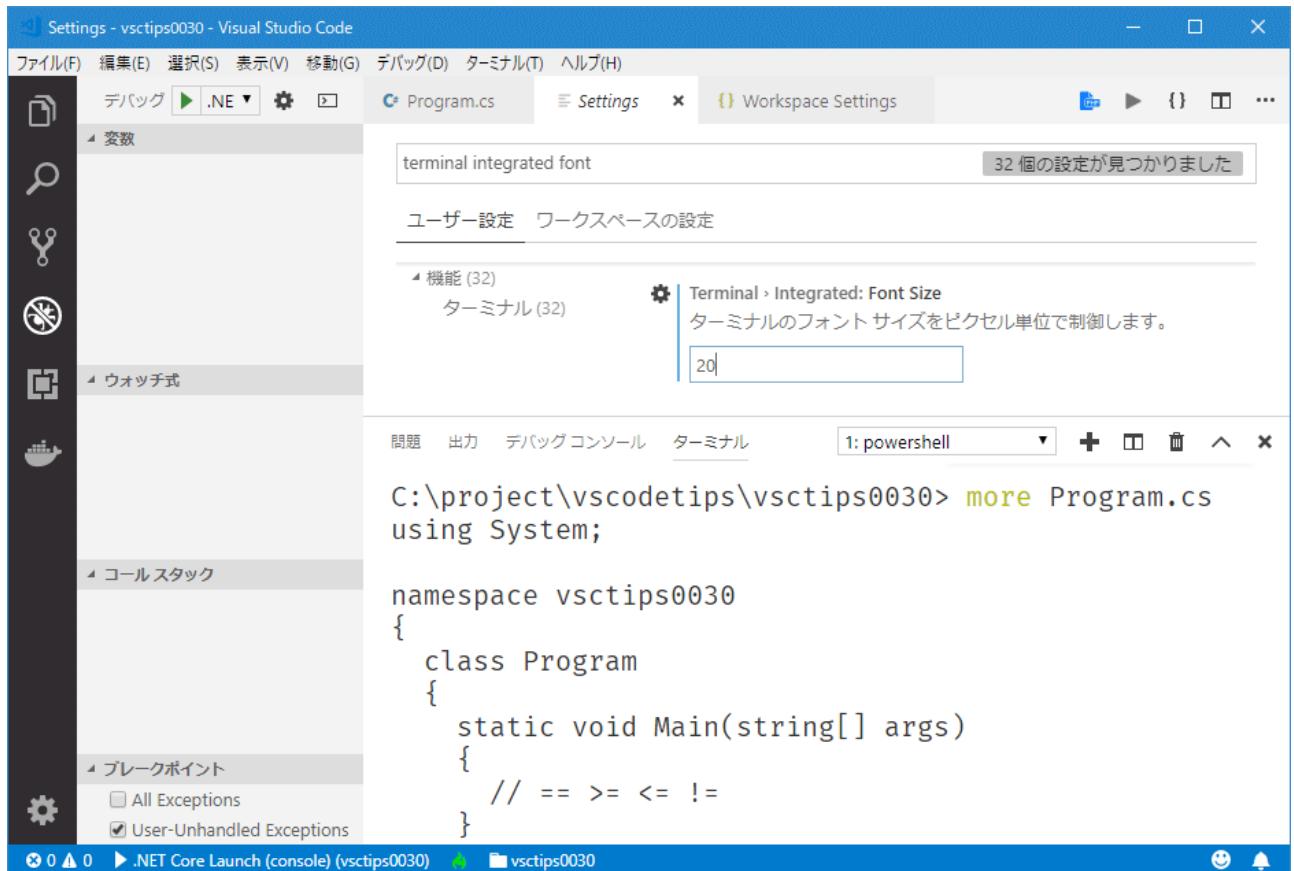
統合ターミナルのフォント関連設定項目

この項目のデフォルトは「""」（空）になっているが、これは `editor.fontFamily` 項目の値に従うこと意味する。統合ターミナルで、エディタと異なるフォントを使用したいときに、この項目で設定するようにしよう。

エディタで使用するフォントに関する設定には、`editor.fontLigatures` 項目があったが統合ターミナルのフォント関連設定には対応するものがない。一方、上の画像の一番下にある `terminal.integrated.fontWeightBold` 項目は統合ターミナルのフォント関連設定にしかない項目となっている。項目名から分かる通り、これは統合ターミナルで使用される太字をどのくらいの太さ（ウェイト）で表示するかを指定するものだ。

統合ターミナルのフォントサイズを指定する

同様に、統合ターミナルで使用するフォントのサイズを指定するには、`terminal.integrated.fontSize` 項目を指定する（ピクセル単位）。デフォルト値は OS によって異なる。以下にフォントサイズを「20」に設定した例を示す。



統合ターミナルのフォントサイズを変更した例

なお、この設定は統合ターミナルが表示されている「パネル」部分の他のタブには影響を与えることには注意しておこう。

VS Code のコマンドラインオプション：起動編

VS Code で、エディタおよび統合ターミナルに表示するテキストのフォントやそのサイズなどを指定する方法を説明する。

Visual Studio Code（以下、VS Code）はコマンドラインから「code」コマンドを実行することでも起動できる。このときにはコマンドラインオプションを指定可能だ。本稿では、それらのうちの幾つかを紹介する。VS Code の拡張機能に関連するコマンドラインオプションもあるが、それらについては別稿で説明する。

オプション	機能
-d/--diff <file1><file2>	<file1>と<file2>を比較
-a/--add <dir>	直前にアクティブとなっていたVS Codeのウィンドウに、指定したディレクトリ<dir>を追加
-g/--goto <file:line[:column]>	fileを開き、lineで指定した行、columnで指定したカラムにカーソルを移動
-n/--new-window	VS Codeの新しいウィンドウを表示する
-r/--reuse-window	既に開いているVS Codeウィンドウを再利用する（そのウィンドウで開いていたフォルダ／ワークスペースは閉じられる）
-w/--wait	このコマンドで開かれたVS Codeのウィンドウが閉じられるまで、コマンドプロンプト／シェルに制御を返さない
--locale <locale>	指定した<locale>を表示言語としてVS Codeを起動
--user-data-dir <dir>	ユーザーデータを保存するディレクトリ<dir>を指定
-v/--version	VS Codeのバージョンを表示
-h/--help	ヘルプを表示

VS Code の主要なコマンドラインオプション

以下では幾つかのコマンドラインについて、もう少し詳しく説明する。ただし、その前に簡単に code コマンドについてまとめておこう。

- カレントディレクトリを VS Code で開く：「code .」コマンドを実行
- 指定したファイルを VS Code で開く：「code <filename>」コマンドを実行
- 以前に開いていたセッションを復元して VS Code を起動する：「code」コマンドを実行

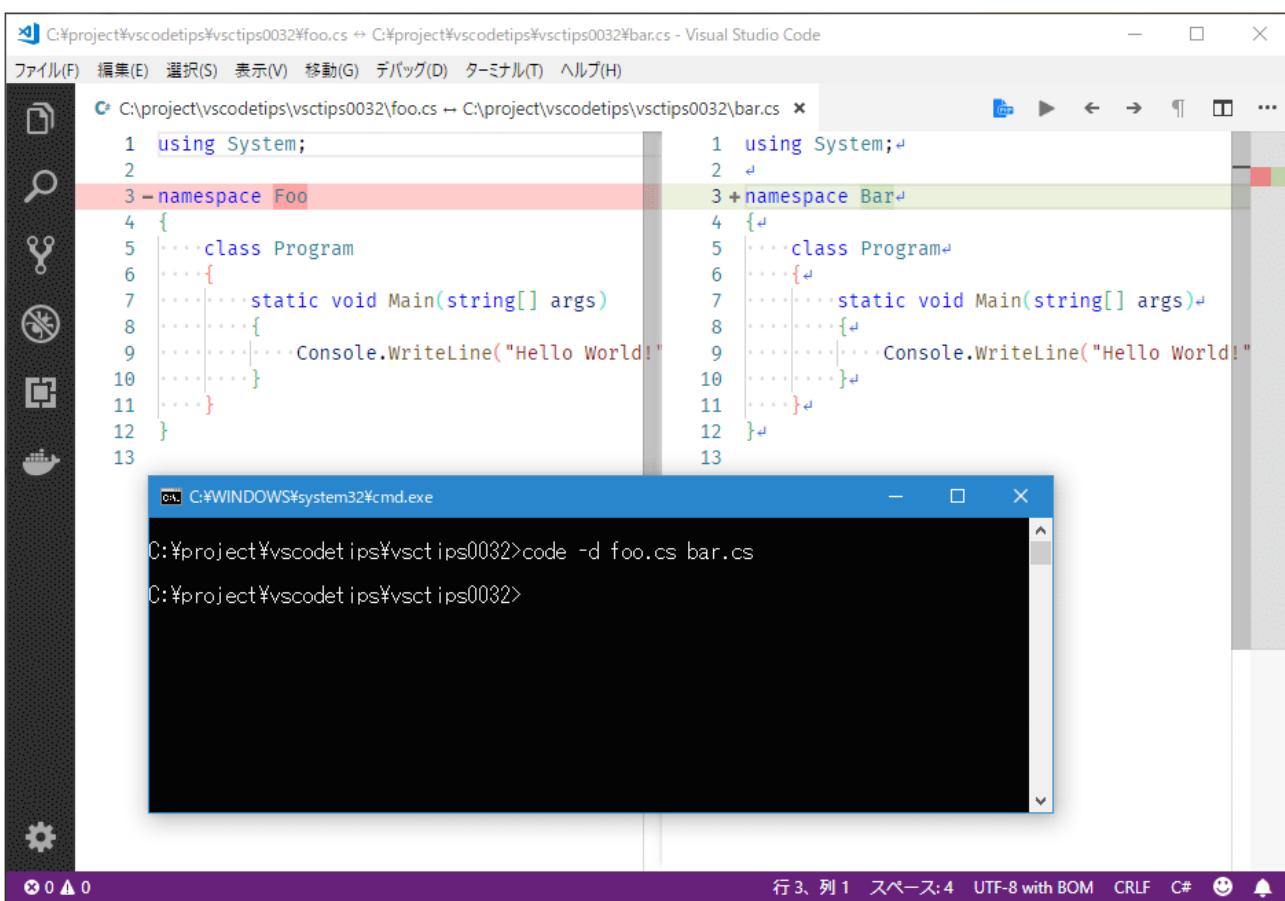
なお、macOS ではコマンドパレットから [シェル コマンド : PATH 内に 'code' コマンドをインストールします] コマンドを実行しておく必要がある。



[シェル コマンド: PATH 内に 'code' コマンドをインストールします] コマンド

-d / --diff : ファイルの差分を表示

VS Code には指定した 2 つのファイルの差分を表示する機能がある。-d / --diff オプションはこれをコマンドラインから利用するためのものだ。オプションに続いて、ファイルを 2 つ指定して、VS Code を起動すると次のようにファイルの差分表示が行われる。以下は「code -d foo.cs bar.cs」コマンドをコマンドラインで実行したところだ。



差分の表示

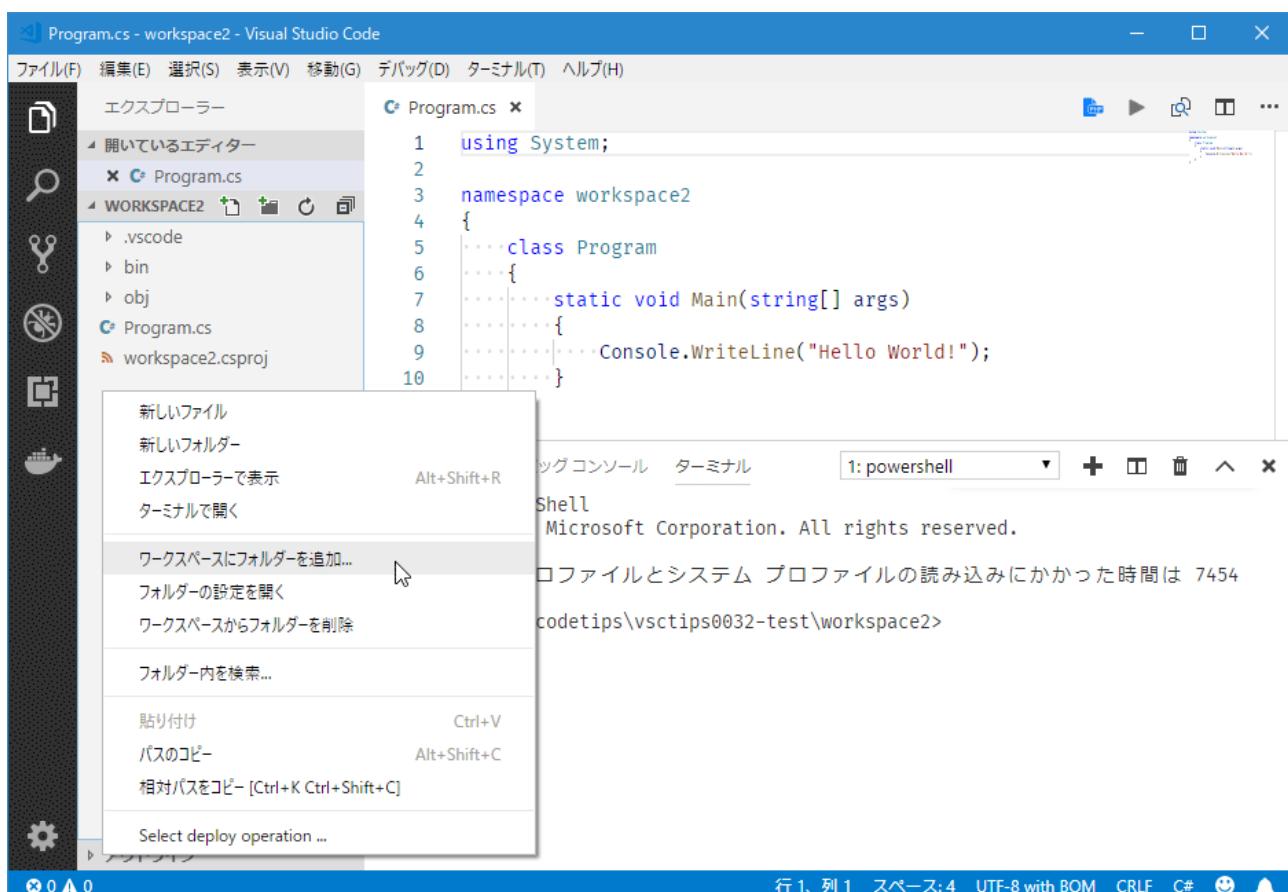
VS Code のウィンドウが既に開かれている場合には、それらのうちのいずれかに差分が表示される。VS Code のウィンドウがない場合には、新たにウィンドウが開かれ、そこに差分が表示される。既にウィンドウがある場合、(恐らくは) VS Code をコマンドラインから実行しようとしたときのカレントディレクトリが VS Code で開かれていれば、そのウィンドウが使われる。そうでなければ、直前にアクティブだったウィンドウが使用される。

VS Code 内でファイルの差分表示を行う方法については「[VS Code でファイルを比較し、差分（diff）を表示するには](#)」を参照してほしい。

-a / --add : ワークスペースにディレクトリを追加

VS Code には「マルチルートワークスペース」という機能がある。この機能を使うと、別々のディレクトリで管理されている複数のプロジェクトをひとまとめのプロジェクトとして扱いたいときに、VS Code 内でそれらを 1 つの論理的なプロジェクトディレクトリのように見なせる。開発の規模が大きなときには役に立つ機能だ。

-a / --add オプションは、VS Code で直前にアクティブになっていたウィンドウに、指定したディレクトリを追加して、マルチルートワークスペース化するものだ。既に開いているウィンドウが対象となる点には注意しよう。よって、これは VS Code の [エクスプローラー] ビューを右クリックして、コンテキストメニューから [ワークスペースにフォルダーを追加] を選択するのと同様な操作をコマンドラインで行うことに相当する。



GUI を利用して、ワークスペースにディレクトリを追加しようとしているところ

そのため、このオプションはコマンドプロンプトやシェルから実行するというよりは、統合ターミナルから現在開いているプロジェクトディレクトリに別ディレクトリを追加するといった場合に使うのがよいだろう。以下に例を示す。

```
using System;
namespace workspace2
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
パーソナル プロファイルとシステム プロファイルの読み込みにかかった時間は 7454 ミリ秒です。
C:\project\vscodetips\vsctips0032-test\workspace2> code --add ..\workspace1
C:\project\vscodetips\vsctips0032-test\workspace2>

統合ターミナルを使って、workspace2 ディレクトリに workspace1 ディレクトリを追加してマルチルートワークスペースにしているところ

ディレクトリを追加しても、マルチルートワークスペースの構成情報を保存する `.code-workspace` ファイルは作成されていないことには注意しよう（タイトルバー や [エクスプローラー] ビューには「未設定」とあることに注目）。

-g / --goto : 指定したファイルの指定した行／カラムに移動

`-g` / `--goto` オプションは「`code --goto <file[:line[:column]]>`」形式で指定したファイル (`file`) の指定した行 (`line`)、カラム (`column`) に移動する。`line` と `column` は省略可能だ。`line` を省略した場合は先頭行に移動する（よって `-g` オプションを指定する意味はない）。`column` を省略した場合は `line` で指定した行の行頭にカーソルを移動する。`column` を指定する場合には `line` も指定する必要がある（「`code -g somefile::5`」としても先頭行の 5 カラム目にカーソルが移動することはない）。

-n / --new-window : VS Code の新しいウィンドウを表示

冒頭で述べたが、オプションや引数なしで単に「`code`」コマンドを実行して VS Code を起動すると、以前のセッションを復元して VS Code のウィンドウが開かれる。すぐに以前の続きをから作業できるので、便利といえば便利だが、全く新しく（空のセッションを）起動したいこともある。そのようなときにはこのオプションを指定すると、何も開かれていない状態で VS Code が開かれる。

なお、筆者が試したところでは、Windows / Linux で既に VS Code ウィンドウが開かれている状態で「code」コマンドを実行すると、新規セッションが開かれ（-n オプション指定と同様な動作）、macOS では既に開いているウィンドウ（のいずれか）がアクティブになったことも書き添えておこう。

-r / --reuse-window : 既に開いているウィンドウを再利用

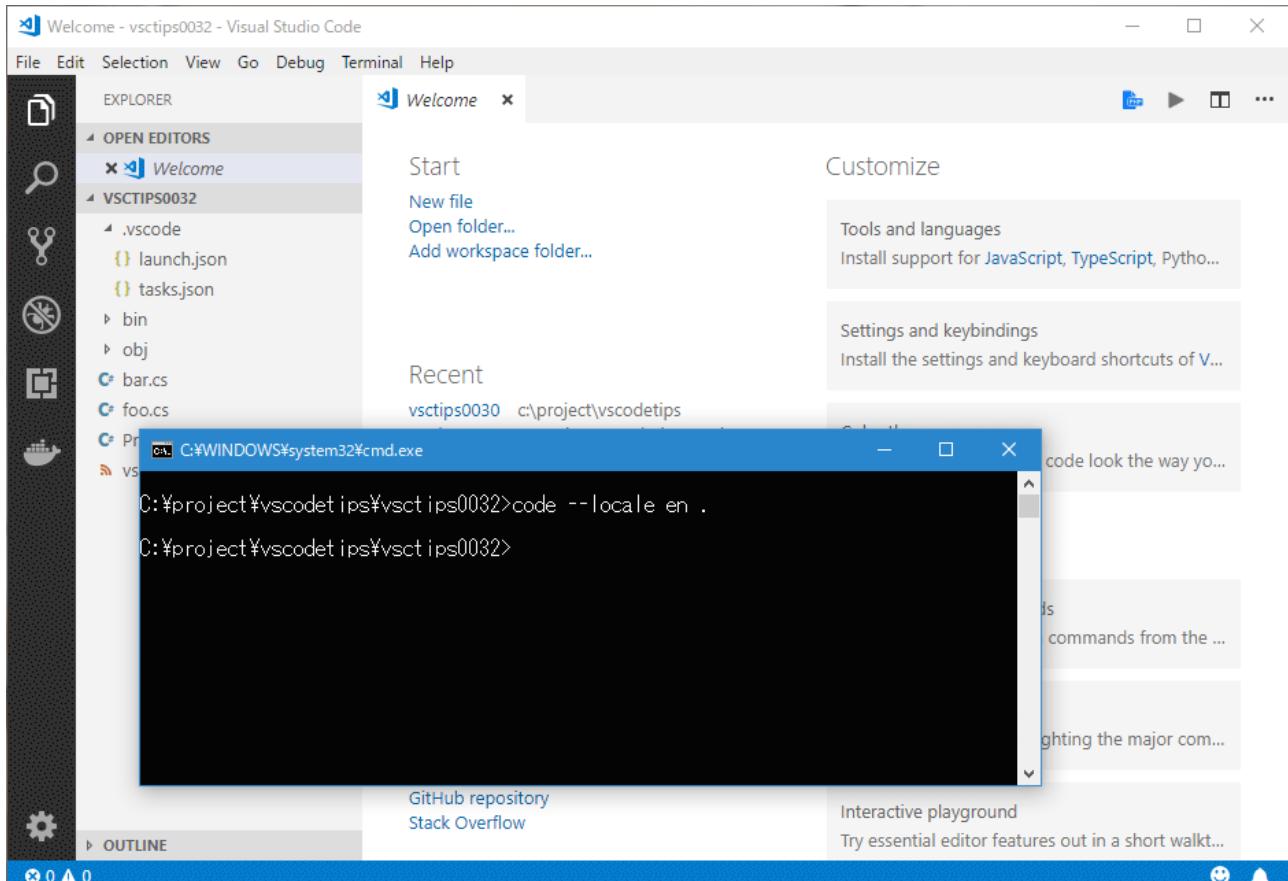
-n / --new-window オプションに対して、既に開かれている VS Code ウィンドウを再利用するためのオプションが -r / --reuse-window オプションとなる。例えば、「code . -r」コマンドを実行すると、直前にアクティブだったウィンドウで開かれていたファイルやディレクトリ、ワークスペースが閉じられて、そのウィンドウにカレントディレクトリが開かれる。

-w / --wait : VS Code ウィンドウ（ファイル）が閉じられるまで待機

-w / --wait オプションは、そのオプションを指定して開いた VS Code ウィンドウやそこで編集していたファイルが閉じられるまで、コマンドプロンプトやシェルに制御を返さないようにするためのものだ。外部ツールと連携しているときには、VS Code での作業が終わるまでは、次のステップに移行できないことがある。そうしたときに使用するオプションだ（そのため、ユーザーがコマンドラインレベルで明示的に指定するというよりは、外部ツールとの連携の中で自動的にこのオプション付きで VS Code のウィンドウが開かれることになるだろう）。

--locale : 表示言語を指定

VS Code では表示言語を切り替えることができる。これには `locale.json` ファイルを編集する方法と、コマンドラインからロケールを指定する方法がある。`--locale` オプションはその後者である。このオプションに続けて、表示言語を指定する。指定可能な値については「[Available locales](#)」を参照のこと。ただし、対応する言語パックが必要になる。以下に「`--locale en`」オプションを指定し、英語を表示言語として VS Code を起動している例を示す。



表示言語を英語にして VS Code を起動したところ

この他にも拡張機能に関連するオプションなどがあるが、それらについては次の項目を参照してほしい。

VS Code のコマンドラインオプション：拡張機能編

VS Code をコマンドラインから起動する場合にはコマンドラインオプションを指定可能だ。その中から拡張機能に関連するものを幾つか紹介する。

Visual Studio Code (以下、VS Code) はコマンドラインから「code」コマンドでも起動できる。このときにはコマンドラインオプションを指定可能だ。本稿では、それらのうちの拡張機能に関連するものを幾つか紹介する。VS Code の起動時に指定する一般的なオプションについては「[VS Code のコマンドラインオプション：起動編](#)」を参照されたい。

なお、ここで紹介するコマンドラインオプションは、拡張機能を無効化して VS Code を起動するオプションを除けば、拡張機能をインストール／アンインストール／一覧するものであり、VS Code ウィンドウを開かずに終了する。

オプション	機能
<code>--install-extension <ext></code>	<ext>で指定した拡張機能をインストールする。 既にインストールされている拡張機能を更新する場合には、--forceを同時に指定する 「識別子@X.Y.Z」形式で特定のバージョンの拡張機能も指定可能 (--forceオプションを指定しなくても強制的にそのバージョンがインストールされる)
<code>--uninstall-extension <ext></code>	<ext>で指定した拡張機能をアンインストールする
<code>--disable-extension <ext></code> <code>--disable-extensions</code>	<ext>で指定した拡張機能または全ての拡張機能を無効化してVS Codeを起動する
<code>--list-extensions</code>	VS Codeにインストール済みの拡張機能を一覧表示する。 --show-versionsを同時に指定すると、拡張機能のバージョンも表示される

VS Code の拡張機能に関連するコマンドラインオプション

以下では幾つかのコマンドラインについて、もう少し詳しく説明する。ただし、その前に簡単に code コマンドについてまとめておこう。

- カレントディレクトリを VS Code で開く：「code .」コマンドを実行
- 指定したファイルを VS Code で開く：「code <filename>」コマンドを実行
- 以前に開いていたセッションを復元して VS Code を起動する：「code」コマンドを実行

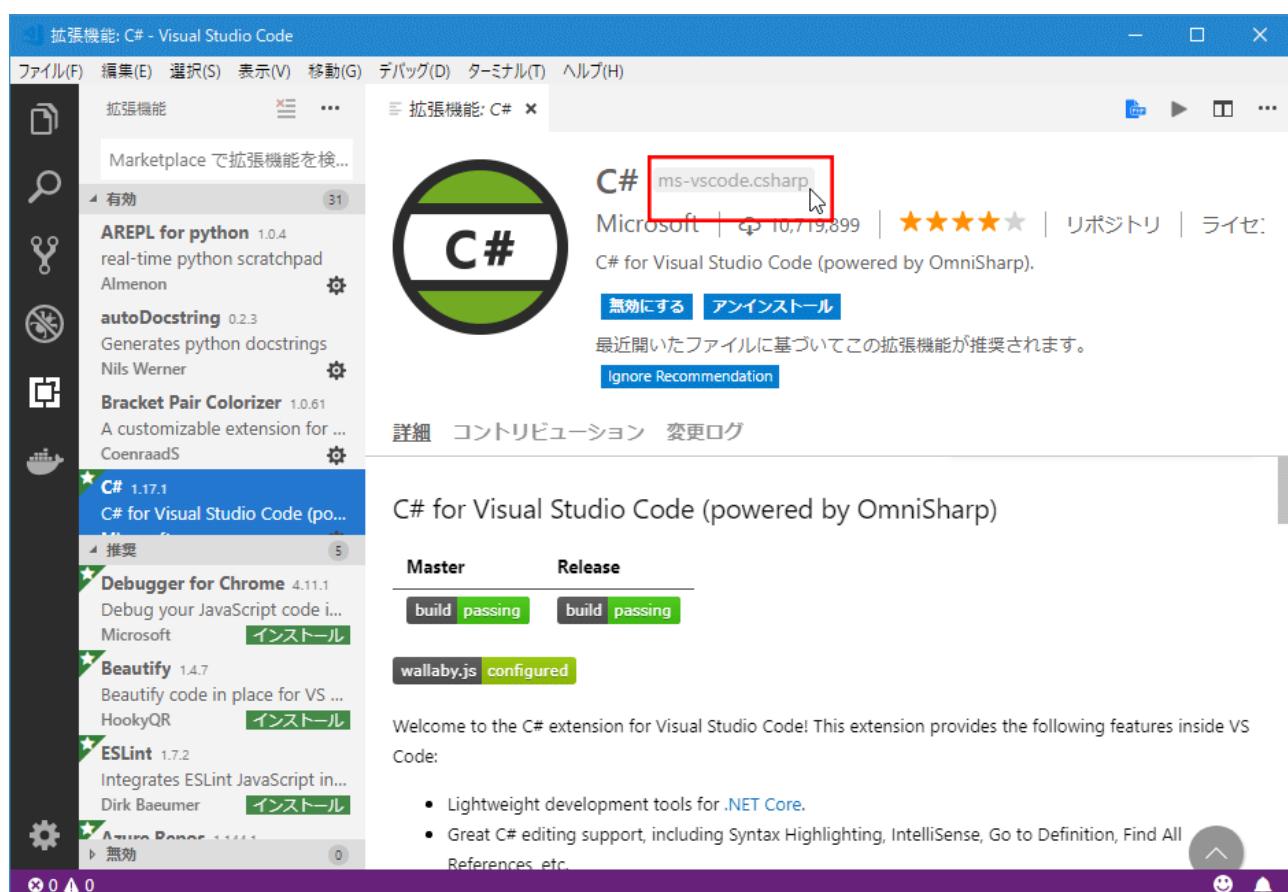
なお、macOS ではコマンドパレットから [シェル コマンド : PATH 内に 'code' コマンドをインストールします] コマンドを実行しておく必要がある。



[シェル コマンド : PATH 内に 'code' コマンドをインストールします] コマンド

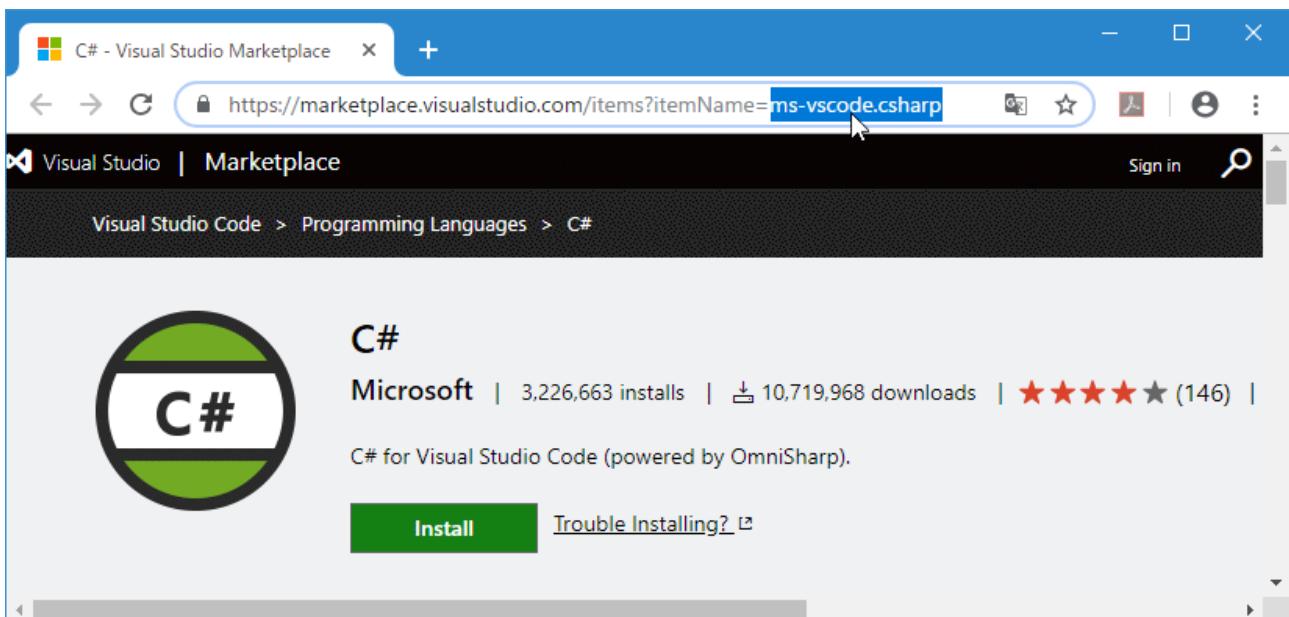
--install-extension : 拡張機能をインストール／更新

コマンドラインから拡張機能をインストールするには、--install-extension オプションに続けて、拡張機能の識別子を指定する。識別子は、VS Code の [拡張機能] ビューで何らかの拡張機能を表示したときに、拡張機能名の隣に表示される。



拡張機能の識別子は、VS Code の [拡張機能] ビューで拡張機能名の隣に表示される（赤枠内）

もしくは、Visual Studio Marketplace でその拡張機能を表示したときにアドレスバーに表示される URL の「=」より後ろの部分となる。



Visual Studio Marketplace では、該当拡張機能の URL に識別子が含まれる（ハイライト表示部分）

例えば、Microsoft 製の C# 拡張機能をインストールするなら「code --install-extension ms-vscode.csharp」コマンドを実行すればよい。

<ext> には識別子以外に、拡張子 VSIX のファイルを直接指定することも可能だ。VSIX 形式のファイルとして拡張機能が配布されている場合には、それをダウンロードして、code --install-extension コマンドに指定できる。例えば、以下は AREPL 拡張機能（Python コードをリアルタイムに評価してくれる拡張機能）の古いバージョンをリポジトリからダウンロードして、インストールしているところだ。

```
C:\WINDOWS\system32\cmd.exe
C:¥project¥vscodet ips¥vsct ips0033>dir
ドライブ C のボリューム ラベルがありません。
ボリューム シリアル番号は D009-3EB2 です

C:¥project¥vscodet ips¥vsct ips0033 のディレクトリ

2018/12/27 09:54    <DIR>      .
2018/12/27 09:54    <DIR>      ..
2018/12/27 09:53        608,522 arepl-0.0.2.vsix
2018/12/27 09:45    <DIR>      helloworld
                  1 個のファイル          608,522 バイト
                  3 個のディレクトリ  24,594,403,328 バイトの空き領域

C:¥project¥vscodet ips¥vsct ips0033>code --install-extension arepl-0.0.2.vsix
Extension 'arepl-0.0.2.vsix' was successfully installed!

C:¥project¥vscodet ips¥vsct ips0033>
```

VSIX 形式で配布されている拡張機能のインストール

なお、このコマンドラインオプションを使って、既にインストール済みの拡張機能を更新しようとしても更新できない点には注意が必要だ。以下は、上でインストールした AREPL 拡張機能の識別子である「almenon.arepl」を指定して「code --install-extension almenon.arepl」コマンドを実行しようとしたところだ。

```
C:\$project\$vscodet ips\$vsct ips0033>dir
 ドライブ C のボリューム ラベルがありません。
 ボリューム シリアル番号は D009-3EB2 です

C:\$project\$vscodet ips\$vsct ips0033 のディレクトリ

2018/12/27 09:54 <DIR> .
2018/12/27 09:54 <DIR> ..
2018/12/27 09:53 608,522 arepl-0.0.2.vsix
2018/12/27 09:45 <DIR> helloworld
      1 個のファイル          608,522 バイト
      3 個のディレクトリ  24,594,403,328 バイトの空き領域

C:\$project\$vscodet ips\$vsct ips0033>code --install-extension arepl-0.0.2.vsix
Extension 'arepl-0.0.2.vsix' was successfully installed!

C:\$project\$vscodet ips\$vsct ips0033>code --install-extension almenon.arepl
Extension 'almenon.arepl' v0.0.2 is already installed, but a newer version 1.0.4
is available in the marketplace. Use '--force' option to update to newer version.

C:\$project\$vscodet ips\$vsct ips0033>
```

--install-extension オプションを指定するだけでは、既にインストールされている拡張機能の更新はできない

英語のメッセージの内容をまとめると「バージョン 0.0.2 がインストールされているが、Visual Studio Marketplace で新しいバージョン 1.0.4 が配布されているので、更新するなら --force オプションを使ってね」と書いてある。そこで、--force オプションを指定すると、次のように新しいバージョンに更新できる。

```
C:\$project\$vscodet ips\$vsct ips0033>dir
 ドライブ C のボリューム ラベルがありません。
 ボリューム シリアル番号は D009-3EB2 です

C:\$project\$vscodet ips\$vsct ips0033 のディレクトリ

2018/12/27 09:54 <DIR> .
2018/12/27 09:54 <DIR> ..
2018/12/27 09:53 608,522 arepl-0.0.2.vsix
2018/12/27 09:45 <DIR> helloworld
               1 個のファイル      608,522 バイト
               3 個のディレクトリ  24,594,403,328 バイトの空き領域

C:\$project\$vscodet ips\$vsct ips0033>code --install-extension arepl-0.0.2.vsix
Extension 'arepl-0.0.2.vsix' was successfully installed!

C:\$project\$vscodet ips\$vsct ips0033>code --install-extension almenon.arepl
Extension 'almenon.arepl' v0.0.2 is already installed, but a newer version 1.0.4
is available in the marketplace. Use '--force' option to update to newer versio
n.

C:\$project\$vscodet ips\$vsct ips0033>code --install-extension almenon.arepl --force
Updating the Extension 'almenon.arepl' to the version 1.0.4
Installing...
Extension 'almenon.arepl' v1.0.4 was successfully installed!
C:\$project\$vscodet ips\$vsct ips0033>
```

更新するには --force オプションも同時に指定してやる

なお、識別子に続けて「@X.Y.Z」形式でバージョンを指定することも可能だ。例えば、AREPL 拡張機能のバージョン 1.0.3 をインストールするには「code --install-extension almenon.arepl@1.0.3」コマンドを実行する。このときには、上で見たような「既に拡張機能がインストールされているので～」というプロンプトを表示して処理を中断することなく、指定したバージョンを強制的にインストールする。インストール済みの拡張機能よりも古いバージョンをインストールしようとしても、警告などは表示されないので注意しよう。

--uninstall-extension : 拡張機能をアンインストール

既にインストールされている拡張機能をコマンドラインからアンインストールするには、`--uninstall-extension` オプションに続けて拡張機能の識別子（もしくは VSIX ファイルのパス）を指定する。以下は上でインストールした AREPL 拡張機能をアンインストールしているところだ。

```
C:\WINDOWS\system32\cmd.exe
ボリューム シリアル番号は D009-3EB2 です
C:\project\vscode\ips\vsct\ips0033 のディレクトリ
2018/12/27 09:54 <DIR> .
2018/12/27 09:54 <DIR> ..
2018/12/27 09:53 608,522 arepl-0.0.2.vsix
2018/12/27 09:45 <DIR> helloworld
    1 個のファイル 608,522 バイト
    3 個のディレクトリ 24,594,403,328 バイトの空き領域

C:\project\vscode\ips\vsct\ips0033>code --install-extension arepl-0.0.2.vsix
Extension 'arepl-0.0.2.vsix' was successfully installed!

C:\project\vscode\ips\vsct\ips0033>code --install-extension almenon.arepl
Extension 'almenon.arepl' v0.0.2 is already installed, but a newer version 1.0.4
is available in the marketplace. Use '--force' option to update to newer version.

C:\project\vscode\ips\vsct\ips0033>code --install-extension almenon.arepl --force
Updating the Extension 'almenon.arepl' to the version 1.0.4
Installing...
Extension 'almenon.arepl' v1.0.4 was successfully installed!

C:\project\vscode\ips\vsct\ips0033>code --uninstall-extension almenon.arepl
Uninstalling almenon.arepl...
Extension 'almenon.arepl' was successfully uninstalled!

C:\project\vscode\ips\vsct\ips0033>
```

拡張機能のアンインストールには `--uninstall-extension` オプションを使用する

既にインストールされている拡張機能は以下で説明する `--list-extensions` オプションで一覧できるので、そこで「アンインストールしたい拡張機能の識別子が何か」の見当が付くだろう（確実を期すなら、上で見たように VS Code の [拡張機能] ビューなどで確認できるが、その場合、コマンドラインでアンインストールする意味があるかは不明だ）。

なお、VSIX 形式のファイルを指定した場合、そのファイルに格納されている拡張機能のバージョンと実際に VS Code にインストールされている拡張機能のバージョンが異なっていてもアンインストールされる。

--disable-extension / --disable-extensions : 拡張機能を無効化

特定の拡張機能を無効化して VS Code を起動するには `--disable-extension` オプション（に続けて拡張機能の識別子）を、全ての拡張機能を無効化して VS Code を起動するには `--disable-extensions` オプションを指定する。後者はオプションの末尾に「`s`」を付けるのを忘れないようにしよう。

以下に AREPL 拡張機能を（もう一度インストールした後に）無効化して VS Code を起動しているところを示す。

```

C:\WINDOWS\system32\cmd.exe
1 個のファイル          608,522 バイト
3 個のディレクトリ   24,594,403,328 バイトの空き領域

C:\$project\$vscodetips\$vsctips0033>code --install-extension arepl-0.0.2.vsix
Extension 'arepl-0.0.2.vsix' was successfully installed!

C:\$project\$vscodetips\$vsctips0033>code --install-extension almenon.arepl
Extension 'almenon.arepl' v0.0.2 is already installed, but a newer version 1.0.4
is available in the marketplace. Use '--force' option to update to newer versio
n.

C:\$project\$vscodetips\$vsctips0033>code --install-extension almenon.arepl --force
Updating the Extension 'almenon.arepl' to the version 1.0.4
Installing...
Extension 'almenon.arepl' v1.0.4 was successfully installed!

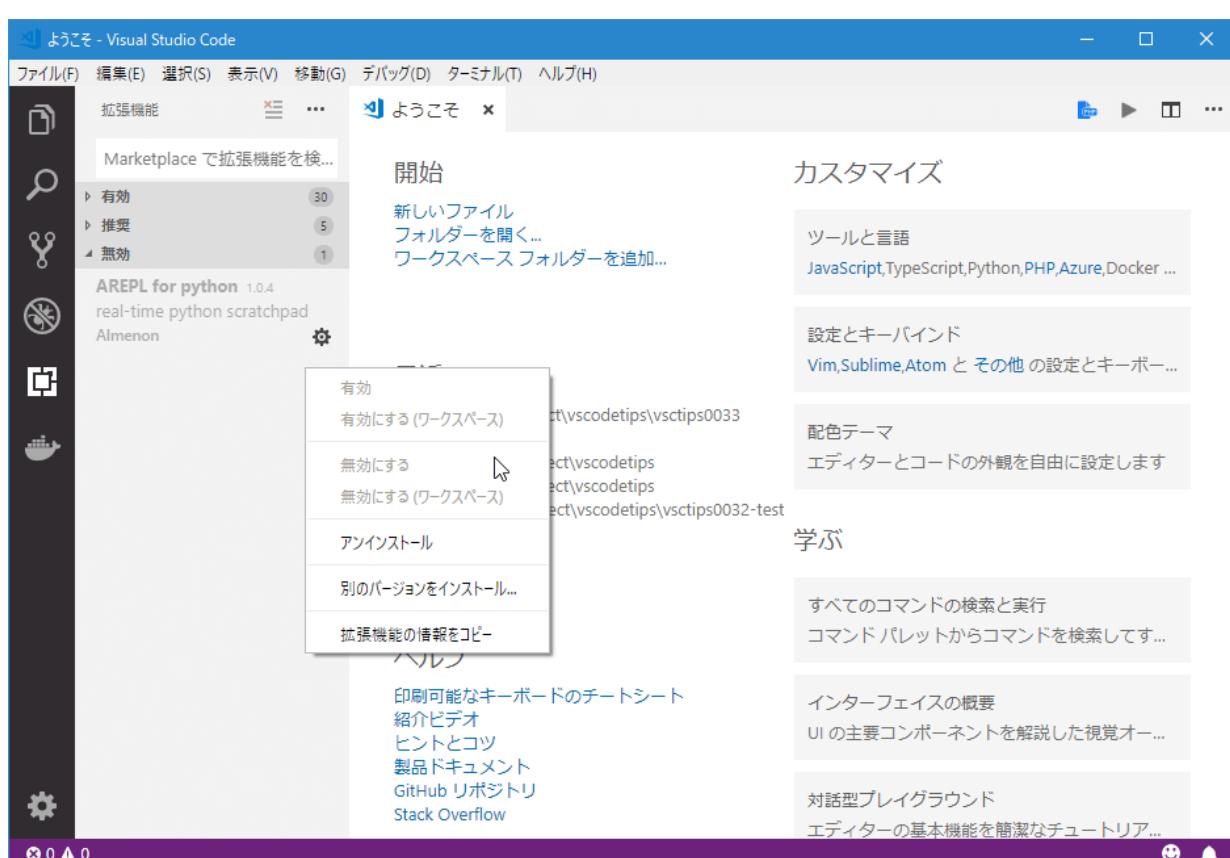
C:\$project\$vscodetips\$vsctips0033>code --uninstall-extension almenon.arepl
Uninstalling almenon.arepl...
Extension 'almenon.arepl' was successfully uninstalled!

C:\$project\$vscodetips\$vsctips0033>code --install-extension almenon.arepl --force
Found 'almenon.arepl' in the marketplace.
Installing...
Extension 'almenon.arepl' v1.0.4 was successfully installed!

C:\$project\$vscodetips\$vsctips0033>code --disable-extension almenon.arepl
C:\$project\$vscodetips\$vsctips0033>

```

「code --disable-extension almenon.arepl」コマンドを実行



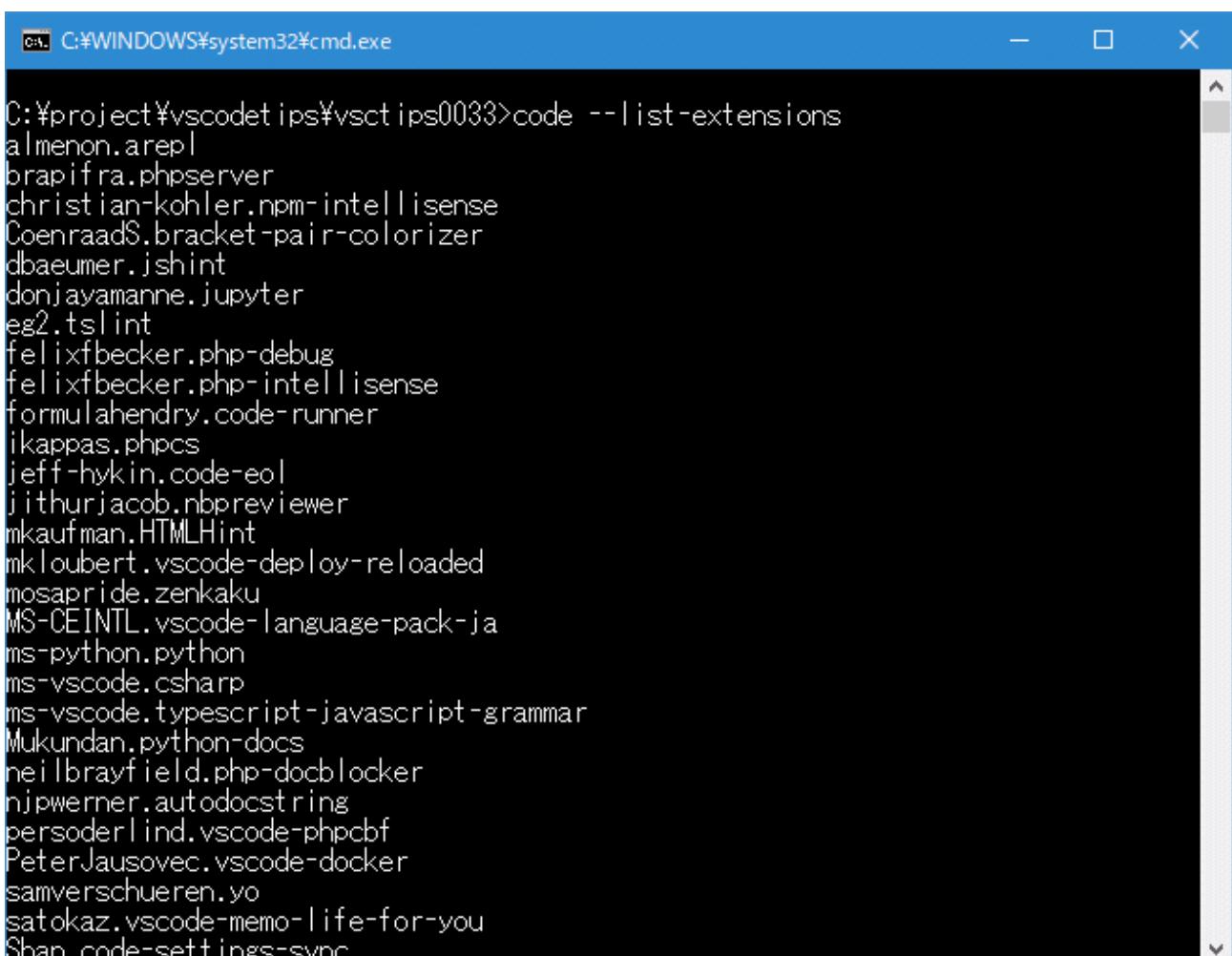
AREPL 拡張機能は無効化され、[拡張機能] ビューの [無効] ページに表示される

拡張機能の無効化の例

コマンドラインで無効化した拡張機能は、[拡張機能] ビューでは [無効] ページに表示される。[拡張機能] ビュー内で拡張機能を無効化した場合には有効化できるが、コマンドラインから無効化した場合には VS Code 内では有効化できない点には注意しよう。

--list-extensions

VS Code にインストールされている拡張機能を一覧するには、`--list-extensions` オプションを指定する。以下に例を示す。



```
C:\$project\$vscode\$ips\$vsct\$ips0033>code --list-extensions
almenon.arepl
brapifra.phpserver
christian-kohler.npm-intellisense
CoenraadS.bracket-pair-colorizer
dbaeumer.jshint
donjayamanne.jupyter
eg2 tslint
felixfbecker.php-debug
felixfbecker.php-intellisense
formulahendry.code-runner
ikappas.phpcs
jeff-hykin.code-eol
jithurjacob.nbpreviewer
mkaufman.HTMLHint
mklobert.vscode-deploy-reloaded
mosapride.zenkaku
MS-CEINTL.vscode-language-pack-ja
ms-python.python
ms-vscode.csharp
ms-vscode.typescript-javascript-grammar
Mukundan.python-docs
neilbrayfield.php-docblocker
njpwerner.autodocstring
persoderlind.vscode-phpcbf
PeterJausovec.vscode-docker
samverschueren.yo
satokaz.vscode-memo-life-for-you
Shan.code-settings-sync
```

VS Code にインストールされている拡張機能一覧

--list-extensions オプションと同時に --show-versions オプションを指定すると、インストールされている拡張機能のバージョンも表示される。以下に例を示す。

```
C:\$project\$vscodet ips\$vsct ips0033>code --list-extensions --show-versions
almenon.arepl@1.0.4
brapifra.phpserver@2.4.6
christian-kohler.npm-intellisense@1.3.0
CoenraadS.bracket-pair-colorizer@1.0.61
dbaeumer.jshint@0.10.20
donjayamanne.jupyter@1.1.4
eg2 tslint@1.0.42
felixfbecker.php-debug@1.12.6
felixfbecker.php-intellisense@2.3.10
formulahendry.code-runner@0.9.5
ikappas.phpcs@1.0.5
jeff-hykin.code-eol@0.3.7
jithurjacob.nbpreviewer@1.2.2
mkaufman.HTMLHint@0.5.0
mkloubert.vscode-deploy-reloaded@0.86.0
mosapride.zenkaku@0.0.3
MS-CEINTL.vscode-language-pack-ja@1.30.2
ms-python.python@2018.12.1
ms-vscode.csharp@1.17.1
ms-vscode.typescript-javascript-grammar@0.0.47
Mukundan.python-docs@0.8.3
neilbrayfield.php-docblocker@1.7.0
njpwerner.autodocstring@0.2.3
persoderlind.vscode-phpcbf@0.0.8
PeterJausovec.vscode-docker@0.4.0
samverschueren.yo@0.9.3
satokaz.vscode-memo-life-for-you@0.4.9
Shan.code-settings-sync@3.2.4
```

VS Code にインストールされている拡張機能一覧（バージョン付き）

拡張機能関連のコマンドオプションには、これらの他にも拡張機能開発時に使用するもの（仕様策定中の API の有効化、デバッグやプロファイリングなどに使用するもの）もあるが、これらについては本稿では省略する。



編集:@IT 編集部
発行:アイティメディア株式会社
Copyright © ITmedia, Inc. All Rights Reserved.