Visual Studio

Visual Studio Code 1st Step Guide

フル バージョン

2015 年 7 月 17 日 (v. 1.0)

Microsoft

このドキュメントは現状版として提供されます。このドキュメントに記載されている情報や見解 (URL 等のインターネット Web サイトに関する情報を含む) は、将来予告なしに変更されることがあります。このドキュメントに記載された例は、説明のみを目的とした架空のものです。実在する事物とは一切関連ありません。内部的な参照目的に限り、このドキュメントを複製して使用することができます。

このドキュメントは、情報提供の目的にのみ提供されるものであり、マイクロソフトはこのドキュメントおよびその記載事項について、明示と黙示とを問わずなんら保証を行いません。

Microsoft、System Center、Visual Studio、および Windows Live は Microsoft グループ各社の商標です。その他、記載されている会社名および製品名は、各社の商標です。

最終更新日: 2015 年 7 月 17 日

1. 目次

1. Visual Studio Code の概要

1.1. Visual Studio Code の登場とその背景

近年スマートフォンや IoT デバイスなどが急速に普及し、それに伴ってクラウドも多種多様に拡大しています。様々なプラットフォームが存在する中で、それぞれに向けたアプリケーションの開発を効率的に行うには、プラットフォームに依存しない開発ツールが重要であると Microsoft は考えています。統合開発環境である Visual Studio も、クロス プラットフォーム開発を強化し、PC のデスクトップアプリケーションやクライアントサーバーアプリケーションだけでなく、モバイルアプリやクラウドのアプリも開発できるようになっています。さらに.NET もオープン化し、開発ツールのカバー範囲を広げています。

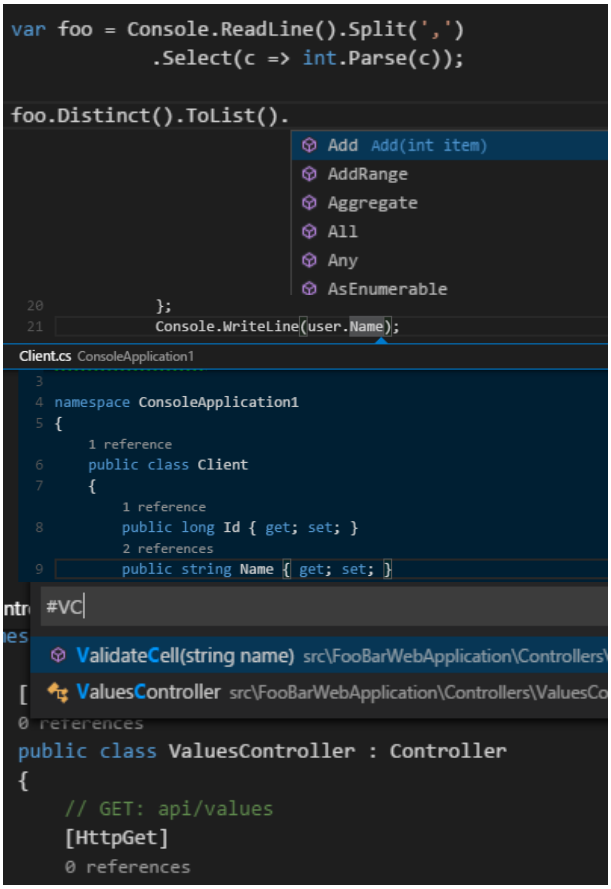
Visual Studio の開発ビジョンである「Visual Studio 1 つですべての開発者があらゆるアプリを開発」を実現するために、Windows 上で稼働する最も完成された統合開発環境として Visual Studio を、また Windows だけでなく Mac や Linux での開発においても利用できるチーム開発基盤・リポジトリ・アジャイル開発基盤として Visual Studio Online を、そして Mac や Linux での軽量かつ高速な高機能エディタとして Visual Studio Code をリリースしました。

1.2. 軽量かつ高機能なエディタ

Visual Studio Code は単にマルチプラットフォームで動作するエディタというだけではなく、Visual Studio が備える高度な機能を搭載しています。例として、従来の高機能エディタが搭載しているような「シンタックスハイライト」や「対応括弧の強調/移動」「マルチカーソル」機能はもちろん、対応言語においては、通常の単語補完より高機能な補完が可能となる「IntelliSense」や型/フィールドの定義や参照などをインラインで表示する「Peek」表示など、IDE と同等の機能を提供します。
いくつかの代表的な機能と概要を以下でご紹介します。

・IntelliSense

入力されたコードの構文と意味を解析して、高機能な入力支援を行う機能です。
ユーザーの入力に追従して柔軟に入力支援内容を変え、快適なコーディングをサポートします。



・Peek

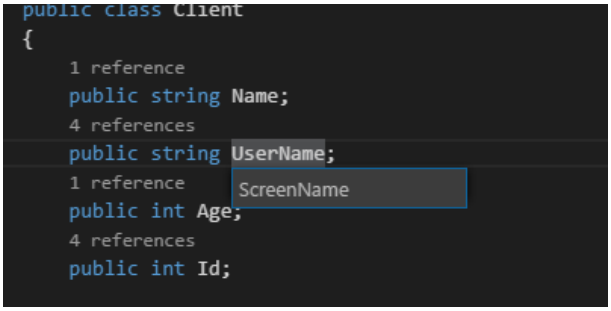
コード中で使用しているフィールドやクラスの定義、およびその参照箇所を複数のソースファイルを横断して検索することができます。検索結果はインラインで表示することができます。

・シンボルの検索

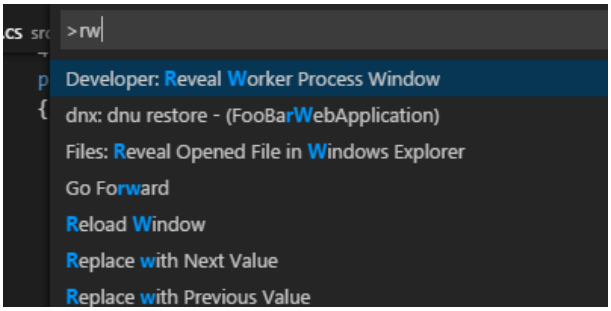
複数のソースファイルを横断して、一致する名前空間や型、インターフェイスなどを検索・移動することができます。

・名前の変更

リファクタリング機能を搭載し、コード中の変数の名前の変更などをはじめ、高度なリファクタリング機能に標準で対応しています。



また Visual Studio Code ではユーザーが快適に開発を行えるよう、多くの機能をキーボードのみで呼び出すことができます。その一つとして「コマンドパレット」があります。ファイルの保存からリファクタリングまで、多くの機能にキーボードからアクセスすることができます。



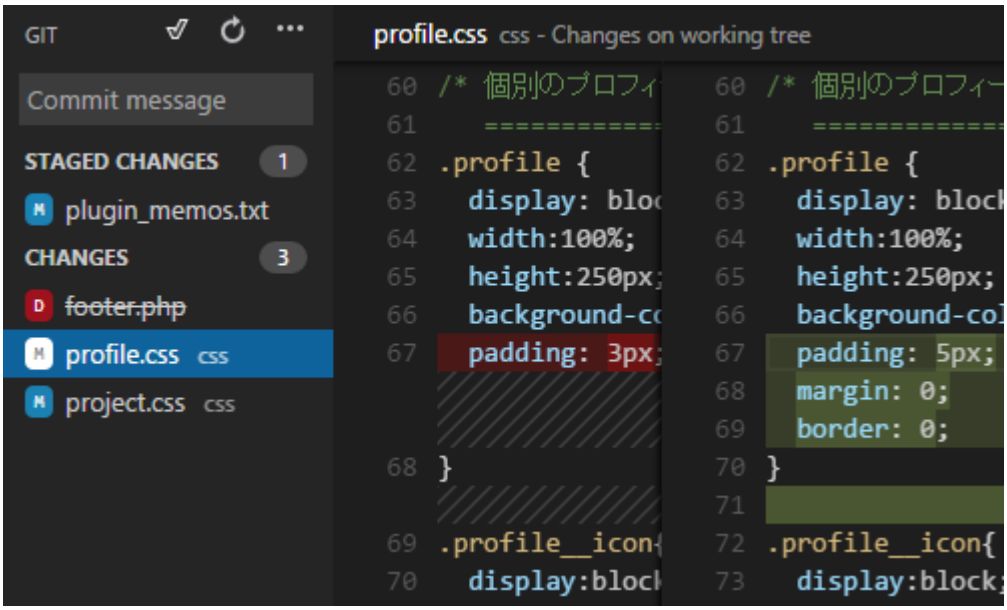
割り当てられているキーバインディングは設定から編集可能であり、ユーザーの好みに合わせて自由に設定することも可能です。

1.3. 様々なユースケースに対応

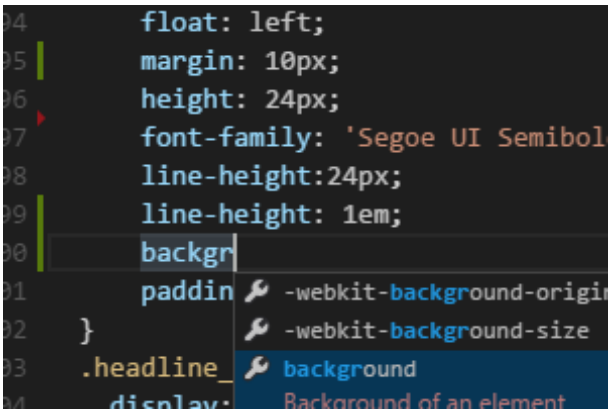
Visual Studio Code はデフォルトで以下の言語に対応しています。

サポート機能	
シンタックスハイライト, 括弧対応付け	Batch, C++, Clojure, Coffee Script, DockerFile, F#, Go, Jade, Java, HandleBars, Ini, Lua, Makefile, Markdown, Objective-C, Perl, PHP, PowerShell, Python, R, Razor, Ruby, Rust, SQL, Visual Basic, XML
（上記の機能に加えて） IntelliSense,エラー表示,アウトライン	CSS, HTML, JavaScript, JSON, Less, Sass
（上記の機能に加えて） リファクタリング,すべての参照の検索	C#, TypeScript

またバージョン管理システムとして Git を採用し、エディタ内でコミットやプッシュ、プルなどの作業を完結させることができます。



さらに編集中のファイルが Git 管理下にある場合は、エディタ中で編集された箇所の行番号周辺にインジケータが表示され、コーディング中に、どの部分が変更されたかが一目で把握できるようになっています。



1.4. ASP.NET 5 や Node.js をはじめとする、最新のテクノロジとの強力な連携機能

Visual Studio Code は標準で ASP.NET 5 および Node.js アプリケーションの開発をサポートしています。ユーザーは開発したアプリケーションの起動からデバッグまでの一通りの操作を、エディタを通して行うことができます。

さらに Grunt や gulp などの各種ユーティリティのサポートや、TypeScript の強力な開発支援、また jQuery/ AngularJS/D3.js をはじめ、最新の JavaScript ライブラリのサポートなど、最新の Web テクノロジを用いた開発に必要な機能も多くサポートしています。

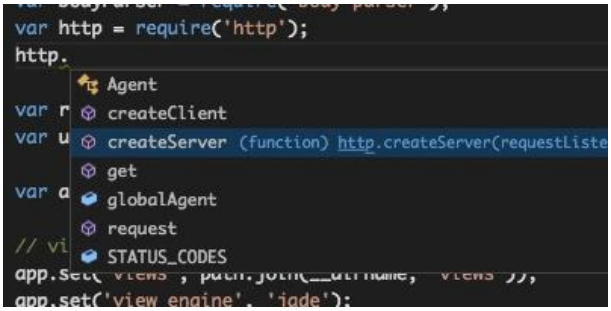
1.5. Visual Studio Code と各種テクノロジの連携

Visual Studio Code は単に高機能なエディタとして使うだけでなく、各種ツールとの連携を行うことにより、ユーザーはより効率的な開発を行うことができます。特に注目すべき点は Node.js、ASP.NET

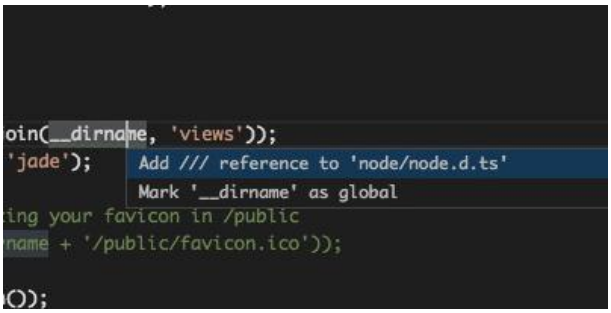
5、TypeScript、そして Git との連携です。ここでは Git を除く 3 つの技術と Visual Studio Code が連携することで可能となる、開発環境をご紹介します。

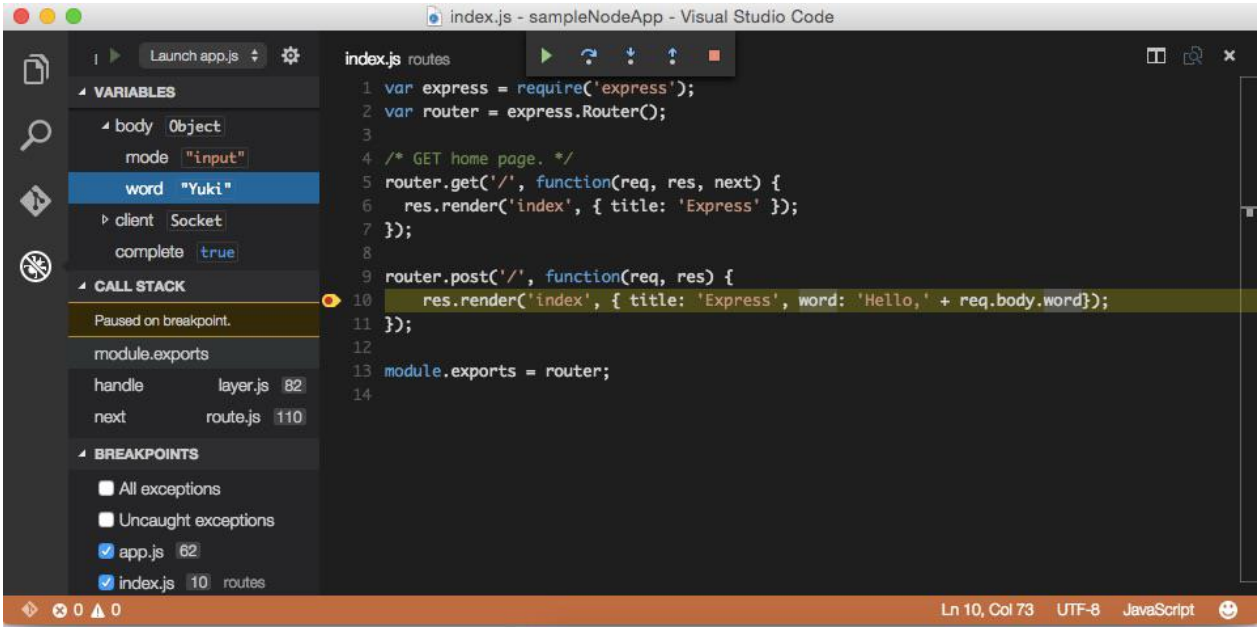
1.5.1. Visual Studio Code と Node.js

IntelliSense は Node.js の開発においても有効に働き、JavaScript/TypeScript での快適なコーディングに大きく貢献します。JavaScript を用いた場合でも右図のように IntelliSense によって型推論が行われ、変数 *http* が http.Server のインスタンスであることから、適切な入力補完が表示されていることが分かります。



また随時問題がないかをリアルタイムに監視し、修正候補がある場合には Code Action が表示されます。右の図では現在のコンテキストにおける __dirname が不明であることを通知し、修正候補を提示していることが分かります。

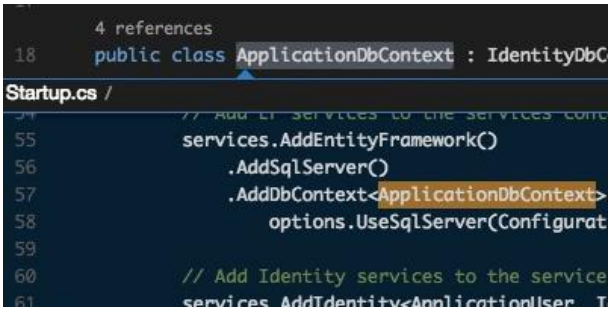




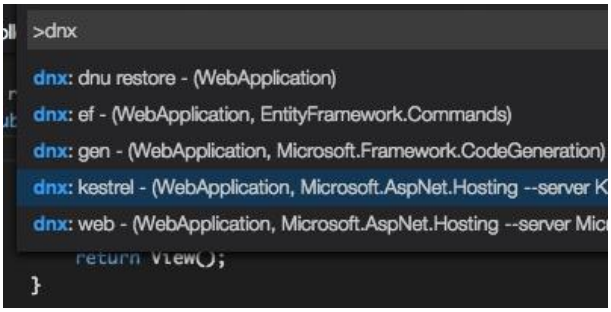
構築したアプリケーションは、エディタから数クリックの操作、もしくは数回のキーボード操作で即座に実行することができます。さらにブレークポイントの設定も簡単に行え、ブレークポイントによって捕捉された時点のプログラムの状態は、変数タブから自由に確認することができます。

1.5.2. Visual Studio Code と ASP.NET 5

C#による開発において IntelliSense は更に有効に機能し、リファクタリングや参照/定義の解決など、Visual Studio Code の持つ、すべての機能が利用可能となります。
例えば右図では *ApplicationDbContext* クラスがどの箇所で用いられているかをソリューション全体から検索し、ピークビューで表示しています。

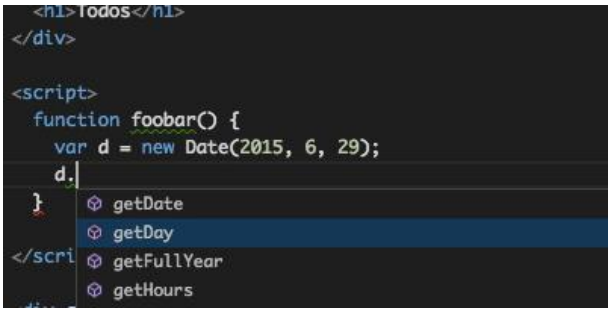


さらに、アプリケーションの実行やパッケージマネージャーの管理も、すべてコマンドパレットから簡単に操作することができます。

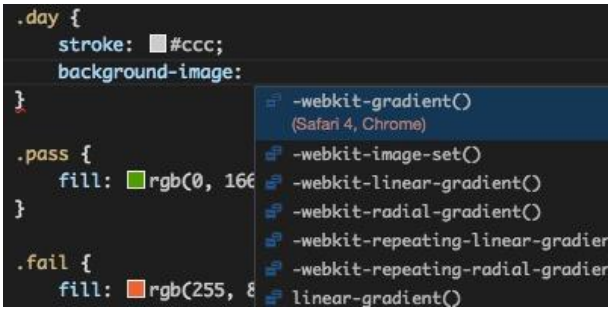


1.5.3. Visual Studio Code と TypeScript

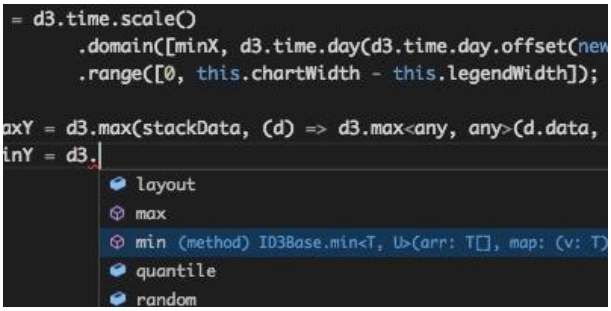
Visual Studio Code は HTML や JavaScript、CSS にも IntelliSense 機能を提供します。右図は HTML 中の *script* 要素で JavaScript を記述しているときの画面です。変数 *d* が Date オブジェクトであることが認識され、適切な補完が行われています。



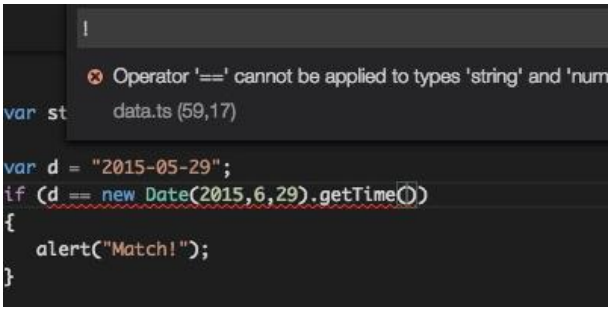
また CSS のコーディング中には、IntelliSense の機能の一部として、入力したカラーのプレビュー表示や、CSS の各種ブラウザ対応状況なども提示します。この他にも、ユーザーに配慮した支援機能が多数用意されており、ユーザーのコーディングによる負担を大きく減らします。



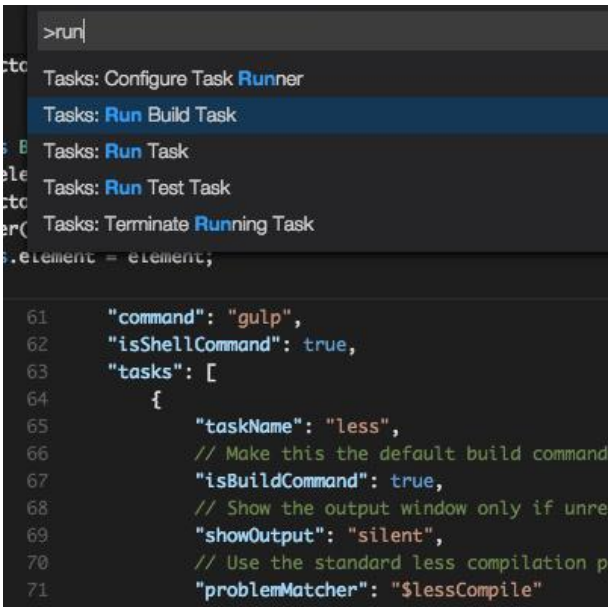
また JavaScript/TypeScript の開発においては、型定義ファイル（d.ts）によって、更に高度な入力補完が提供されます。多くのライブラリの型定義ファイルは GitHub のリポジトリで管理されており、jQuery や AngularJS をはじめ、Web 開発において必要となる様々なライブラリに対応した定義ファイルが用意されています。上図は D3.js を用いた TypeScript アプリケーションの開発を行っている画面です。IntelliSense によって、D3.js が持つ関数や取り得るパラメータ、およびその型が提示されていることが分かります。



さらに入力中のコードはリアルタイムにエラー分析が行われ、型定義ファイルとも合わせて入力されたコードが構文・型情報などの意味的に正しいものであるかを検証し、ユーザーに通知を行います。



TypeScript アプリケーションのコンパイルや各種テストの実行も、Visual Studio Code を通して簡単に行うことができます。右図はコマンドパレットを呼び出して、ビルドを実行する画面です。



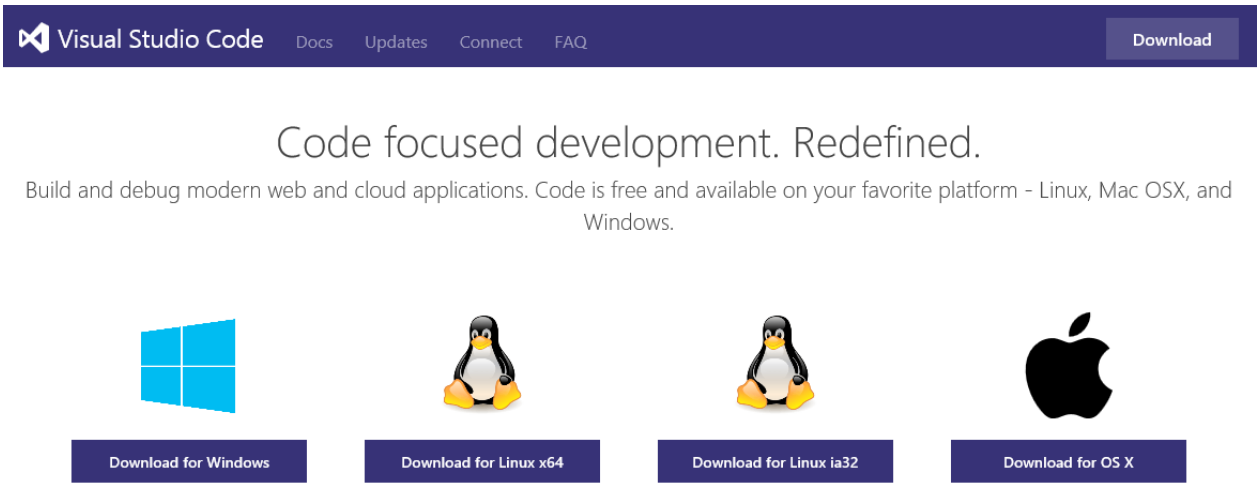
また Grunt や gulp などの各種タスクランナーとの連携も簡単に行うことができます。右図はタスクランナーの設定ファイルに、gulp を用いて less のコンパイルを行う動作を記述している画面です。

2. Visual Studio Code の基本操作

この章では簡単な JavaScript アプリケーションの構築を例に、Visual Studio Code の基本的な操作についてご紹介します。

2.1. Visual Studio Code のインストール

Visual Studio のインストール方法は非常に簡単です。インターネット接続と簡単なコマンド操作ですぐに導入することができます、ここではプラットフォームごとのインストール方法についてご説明します。



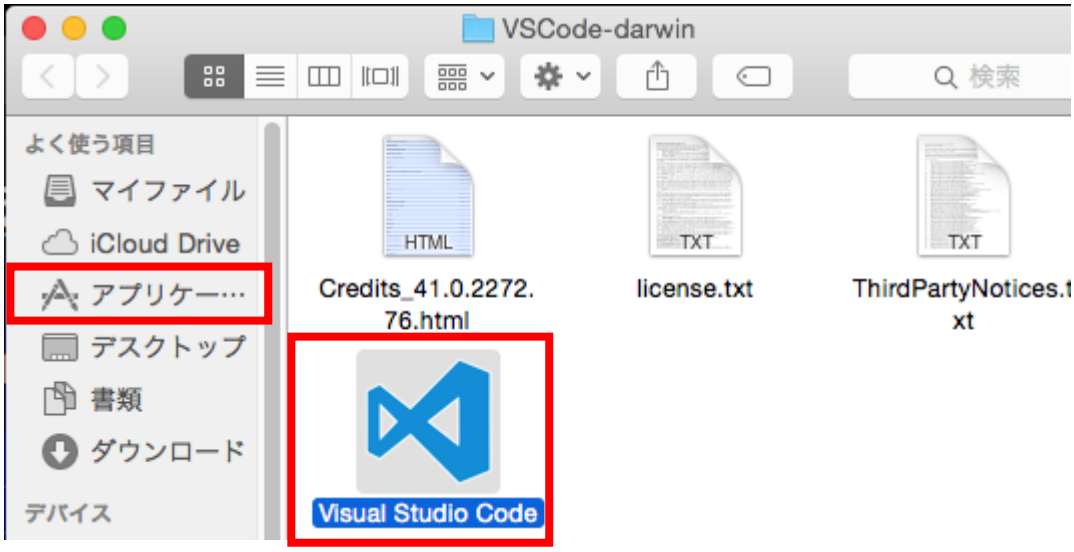
2.1.1. インストール方法（Windows）

1. https://code.visualstudio.com/Download にアクセスします
2. [Download for Windows]ボタンをクリックします
3. インストーラ（VSCodeSteup.exe）のダウンロードが始まります
4. ダウンロードされたインストーラを実行します。自動でインストールが始まります
5. インストール中、数分程度は右のような画面が表示される場合がありますが、そのままウインドウが消えるのをお待ちください。
6. インストールが完了すると Visual Studio Code が起動します

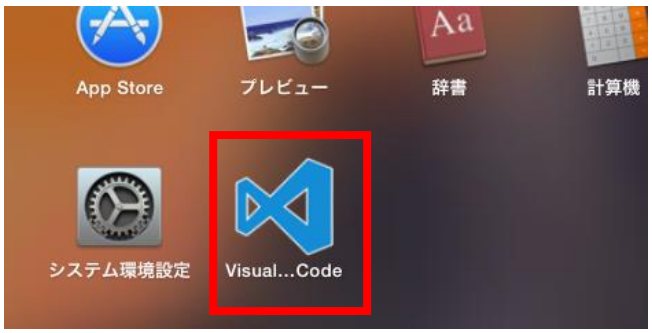
2.1.2. インストール方法（Mac）

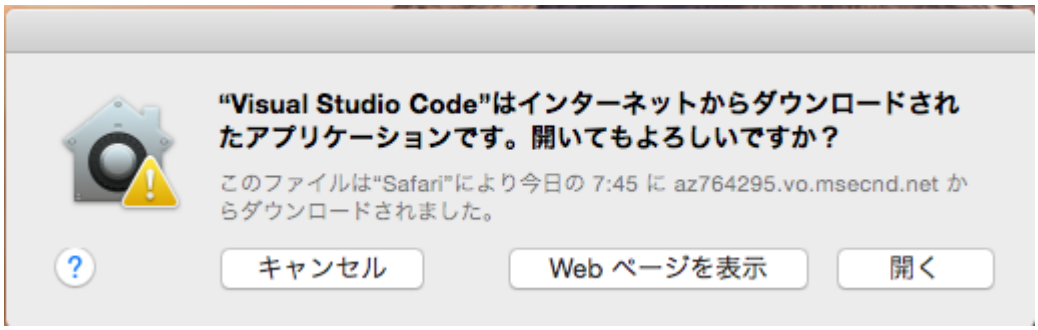
1. https://code.visualstudio.com/Download にアクセスします
2. [Download for OS X]ボタンをクリックします

3. Zip ファイル（VSCode-darwin.zip）のダウンロードが始まります
4. ダウンロードされた Zip ファイルを解凍します。（通常はダブルクリックすると自動で展開されます）
5. 解答されたファイルの中にある[Visual Studio Code.app]を、Finder などを用いてアプリケーションフォルダへコピーします。以下の画面では、右パネルにある[Visual Studio Code]を、左パネルにある[アプリケーション]へドラッグ＆ドロップすることで、コピーすることができます。



6. コピーが完了すると、Launchpad などのランチャーから Visual Studio Code を開くことができます。





このような警告が表示された場合は、[開く]ボタンをクリックします。

7. 必要に応じて、2.1.4 の Tips を実行します。

2.1.3. インストール方法（Linux）

1. https://code.visualstudio.com/Download にアクセスします
2. ご利用のアーキテクチャに合わせて、[Download for Linux x64]か[Download for Linux ia32]ボタンをクリックします
3. Zip ファイル（VSCode-linux-x64.zip もしくは VSCode-linux-ia32.zip）のダウンロードが始まります
4. ダウンロードされた Zip ファイルを解凍します。（unzip コマンドなどを使うと良いでしょう）
5. 解答されたフォルダの中にある[Code]を実行するとアプリケーションを実行することができます。インストールを行うには、例えばパスの通っている場所に Code へのシンボリックリンクを作成すると良いでしょう。

Linux に関するインストールの詳細は Setting up Visual Studio Code を参照してください。

2.1.4. Visual Studio Code インストールの Tips

Visual Studio Code はランチャーから起動させる方法のほか、各種シェルを用いて、ファイル名やフォルダを引数に与えて起動させることができます。この場合、引数に与えたディレクトリやファイルが起動と同時に読み込まれ、ファイルを開く処理などを行うことなく、スムーズに作業を行うことができます。

例として Mac を使用している場合には、以下の行をシェルの設定ファイル（.bash_profile や.zshrc など）に追加してください。

```
code () { VSCODE_CWD="$PWD" open -n -b "com.microsoft.VSCode" --args $* ;}
```

これにより、作業対象のディレクトリやファイルに対して

```
code [FileName/Directory]
```

として Visual Studio Code で開くことができます。

また **Windows を利用している場合**には**デフォルトで設定されています**ので、コマンドプロンプトから上記コマンドを入力することで Visual Studio Code を起動することができます。

Linux や Mac、および使用しているシェルの種類によって設定方法は異なりますので、詳細は [Setting up Visual Studio Code] の Tip を参照してください。

2.2. Visual Studio Code の基本操作

ここでは Visual Studio Code の基本となる操作方法を、簡単な Web アプリケーションの構築を例にご紹介します。
なお、今回作成するアプリケーションのプロジェクトは以下のリンクからダウンロードすることもできます。
https://github.com/bonprosoft/FirstWebSite

2.2.1. ファイルとフォルダおよびプロジェクト

Visual Studio Code はファイル単体を開いて編集を行うこともできますが、フォルダを作業ディレクトリとして開いて、エディタ内でフォルダに含まれるファイルの検索/編集を行うこともできます。

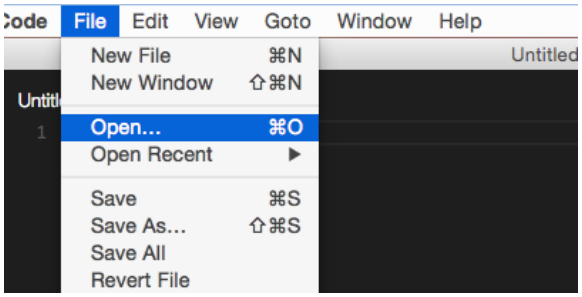
また開いた作業ディレクトリに package.json や project.json、その他各種設定ファイル・ソリューションファイルが含まれていた場合には、自動的にその読み込み、ファイルやフォルダの構成を把握します。これにより、複数のファイルにまたがった参照の検索やプロジェクトに対するタスクランナーなど、ソリューション全体に対する操作を行うことが可能となります。

Web サイトや Web アプリケーションなど、複数のファイル関係が意味を持つ場合にはソースファイル単体を開くのではなく、フォルダを開くことをお勧めします。

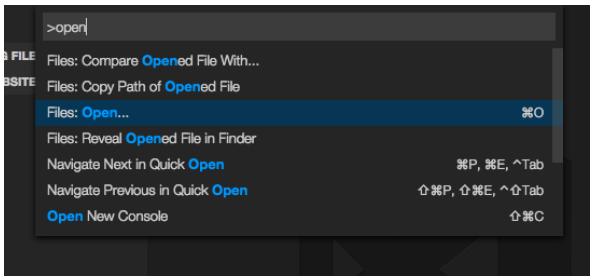
2.2.2. フォルダを開く

では、簡単な Web サイトの制作を行う作業スペースを作成しましょう。以降、画面のスクリーンショットは Mac が基本となりますが、Windows も同様に操作を行うことができます。

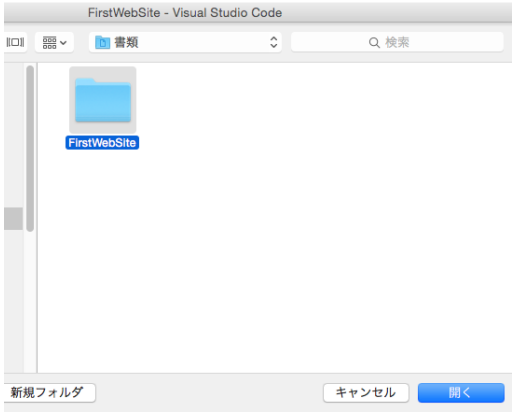
1. [File]メニューから[Open...]を選択します。



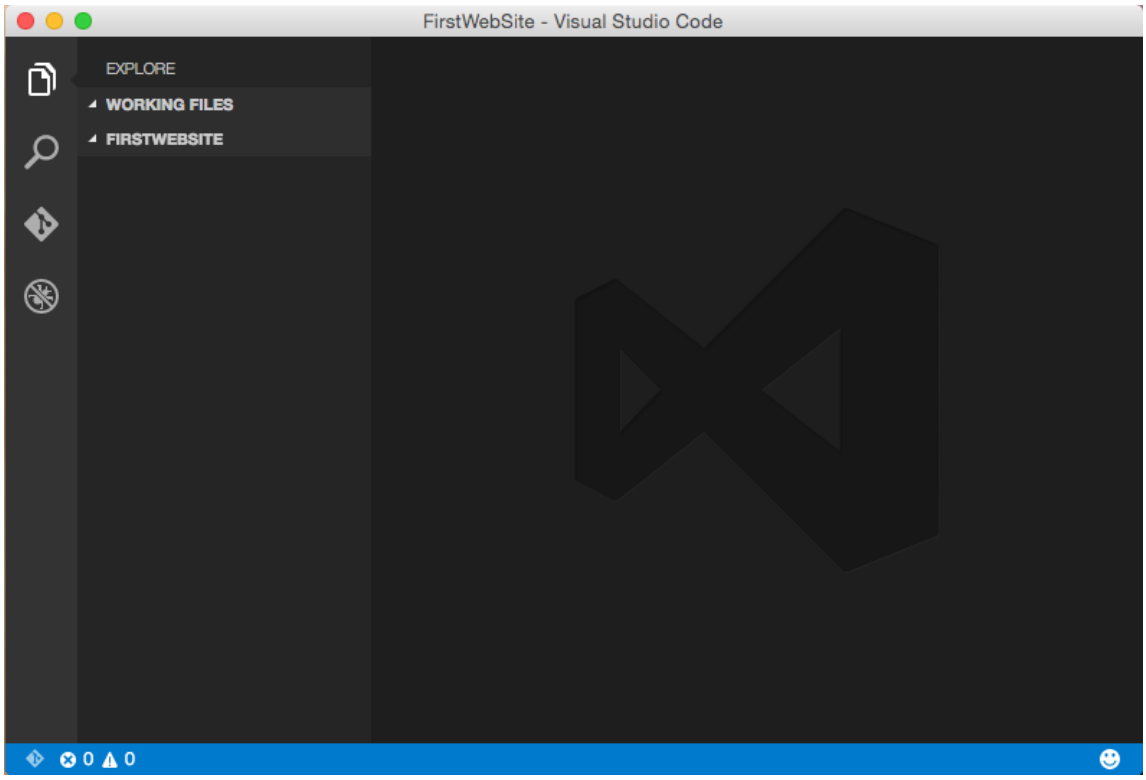
もしくはショートカットである⌘O（Windows は Ctrl+O）キーを同時に押すか、⇧⌘P（Windows は Shift+Ctrl+P）キーを同時に押してコマンドパレットを表示し、[Files: Open…]を選択することで同様のウインドウを表示することができます。



2. 任意の名前のフォルダ（ここでは *FirstWebSite* とします）を作成し、作成したフォルダを選択して[開く]をクリックします。



3. 選択したフォルダが作業ディレクトリとして読み込まれます。

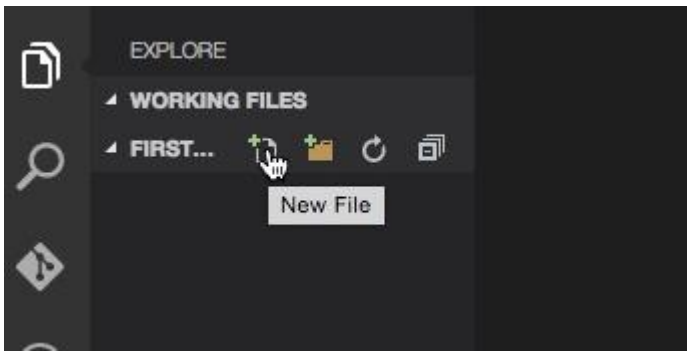


以上で作業ディレクトリを読み込むことができました。

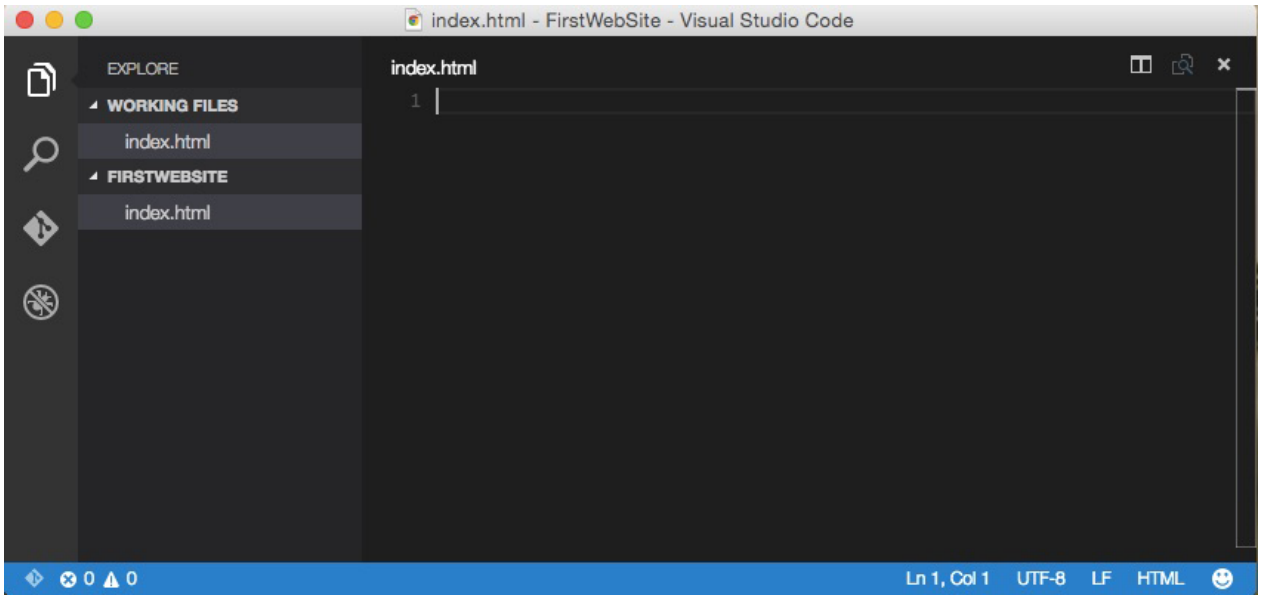
2.2.3. ファイルを新規作成/追加する

では作業ディレクトリにファイルを新規作成しましょう。この操作もコマンドパレット/ショートカットからすべて行うことができます。

1. 画面左側の[作業ディレクトリ名]（ここでは FIRSTWEBSITE となっています）にマウスをホバーさせると、フォルダに対するアクションボタンが表示されます。
2. 一番左側のボタン[New File]をクリックします。



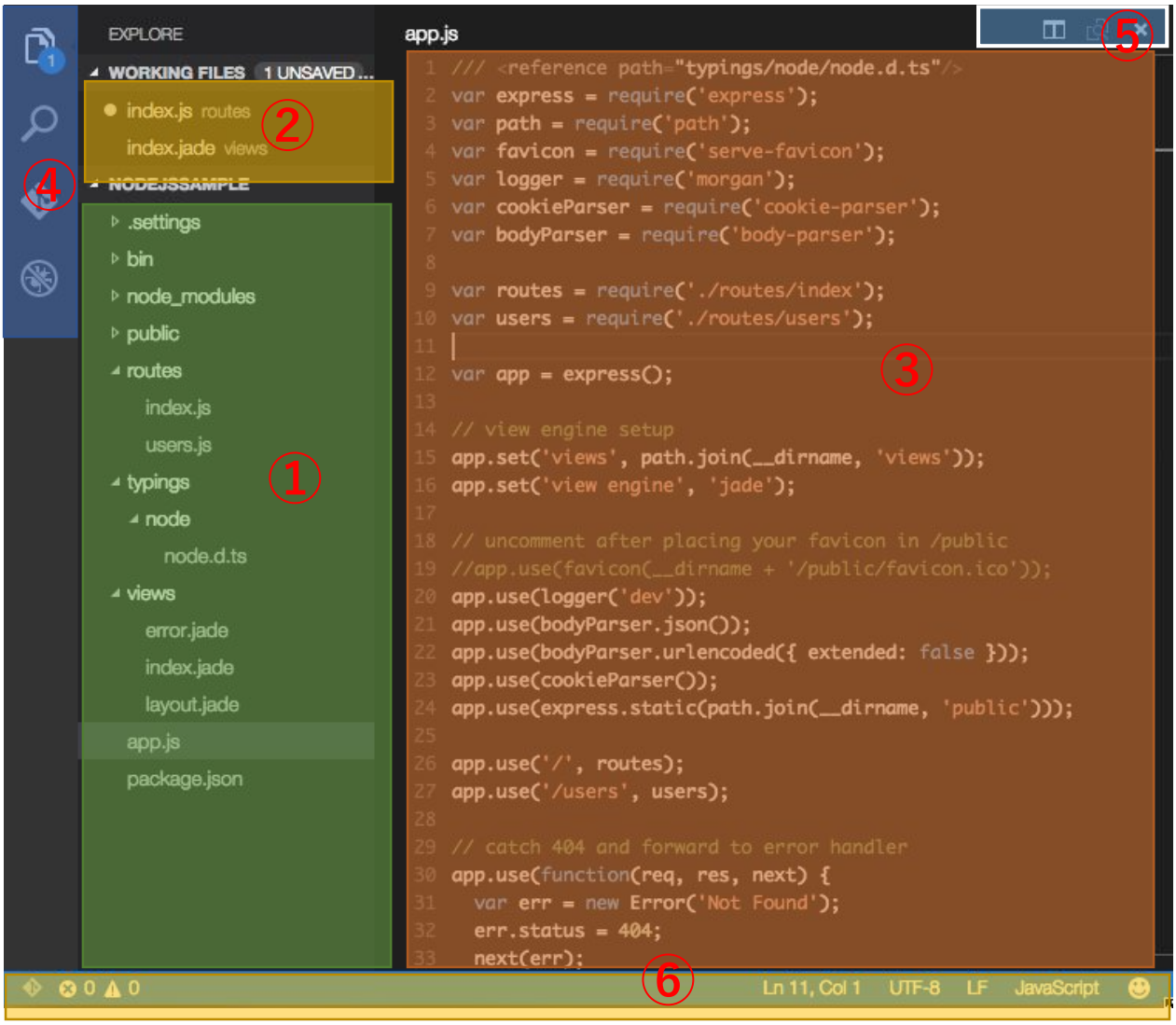
3. ファイル名として *index.html* を入力し、Enter キーを押します。



以上の操作を行うと、エディタに index.html の編集画面が表示されることが確認できます。

2.2.4. Visual Studio Code の基本的なレイアウト

コーディングに移る前に、Visual Studio Code の画面構成について簡単にご説明します。



① 作業ディレクトリを表示します。 （**サイドバー**）

② 現在開いているファイル一覧を表示します。○マークがついているものは、保存されていない変更があることを意味します。（**サイドバー**）

③ エディタを表示します。（**エディタ**）

④ 現在動作中のビューレットの表示と切り替えを行えます。上から順に、「エクスプローラ」「検索」「Git」「デバッグ」を意味します。（**ビューバー**）

⑤ エディタ画面の分割やプレビューなど、エディタに関するアクションを行えます。

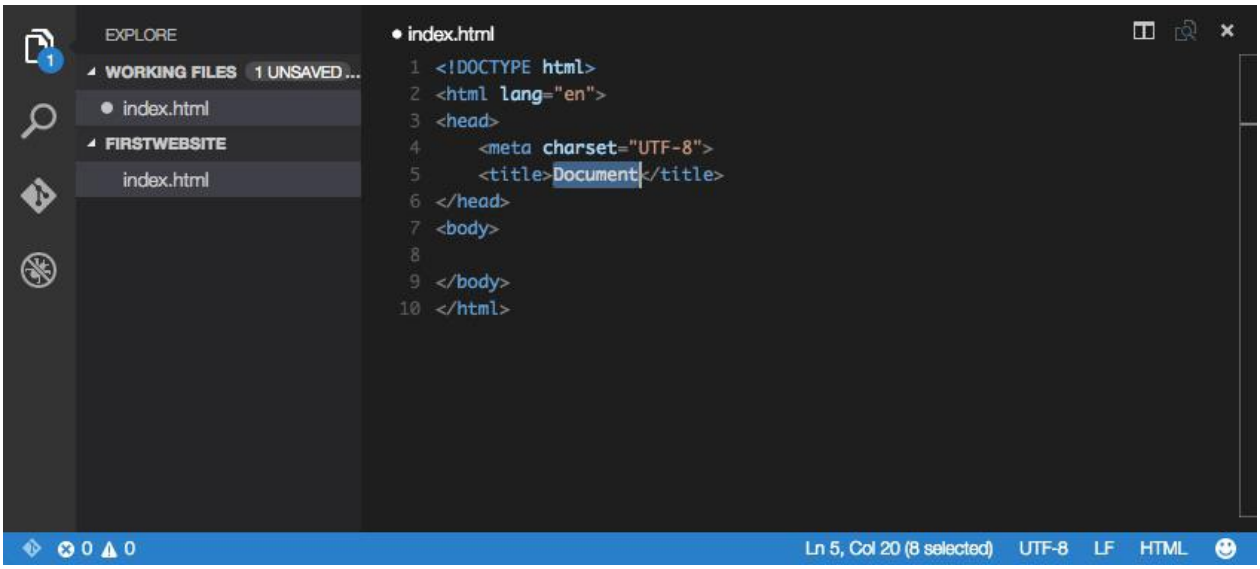
⑥ 左側にはエラー表示や Git 連携状況などロード中のプロジェクトに関する内容、右側には現在編集中のファイルに関する情報やエディタの動作状況を表示します。（**ステータスバー**）

2.2.5. Visual Studio Code を用いた初めての開発

では作成した index.html に対して実際にコードを書いていきましょう。
エディタに以下のコードを入力し、”5”を入力し終えた時点で **Tab キー**を押します。

```
html:5
```

するとエディタに HTML5 のスニペットが展開され、Title にフォーカスが当たります。



Title 要素の値（上の画像では”Document”）を任意の値に変更し、body の中に以下のコードを入力します。

```
<h1>Hello!</h1>
ul>li*5
```

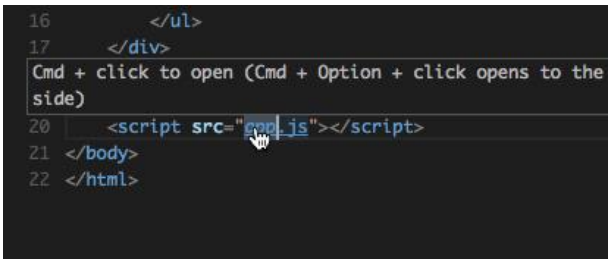
再び **Tab キー**を押すと、今度は li 要素が 5 つ含まれる ul 要素が展開されます。

このように Visual Studio Code の HTML/CSS/Less/Sass では、Emmet によるスニペットを利用することができます。その他 Emmet に関する詳細は、 Emmet cheat seat をご覧ください。

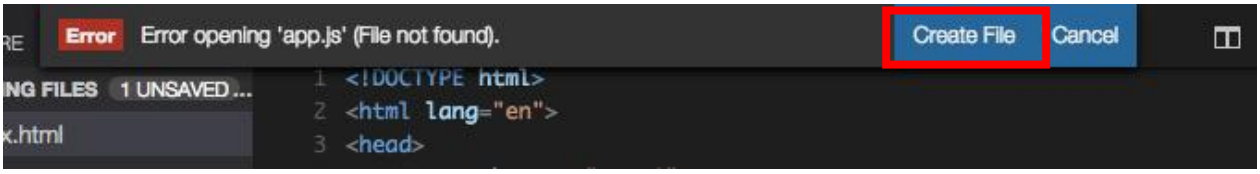
続いて body 要素内に以下のコードを入力します。

```
1 <input type="button" value="Click!" id="rotate"></input>
2 <script src="http://ajax.aspnetcdn.com/ajax/jQuery/jquery-1.11.3.min.js"></script>
3 <script src="app.js"></script>
```

⌘キー（Windows では Ctrl キー）を押しながら、上記コード 2 行目の app.js をクリックします。（右図参照）



するとエディタ上部に「app.js が見つからない」とエラーが表示されますので、[Create File]ボタンを押します。



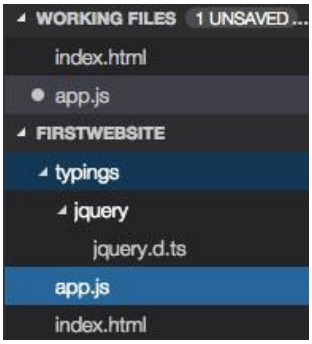
これにより自動的に app.js が作成され、作業ディレクトリに app.js が追加されていることが確認できます。
続いて、app.js に以下のコードを入力します。

```
1 $('#rotate').on('click', function () {
2
3 });
```

上記コードの 1 行目の$に注目すると、緑色の波線が表示されていることがわかります。ここにカーソルを合わせると、右図に示す電球マークが表示されます。この電球マークはエラーなどが見つかった際、ユーザーに解決策を提案出来る時に表示されます。

電球マークをクリックし、"Add /// reference to jquery/jquery.d.ts"を選択します。これにより app.js の 1 行目に TypeScript のリファレンスコメントが挿入されたことが分かります。

更に、この操作により Visual Studio Code が、DefinitelyTyped リポジトリから jQuery の TypeScript の型定義ファイルを自動でインポートし、IntelliSense を jQuery に対応させることができます。画面左側のサイドバーには、新しく typings フォルダが作成されており、型定義ファイルも自動的に格納されていることがわかります。



また先ほどまで表示されなかった IntelliSense も、正常に機能していることがわかります。

では、app.js のコードを以下の通り修正しましょう。

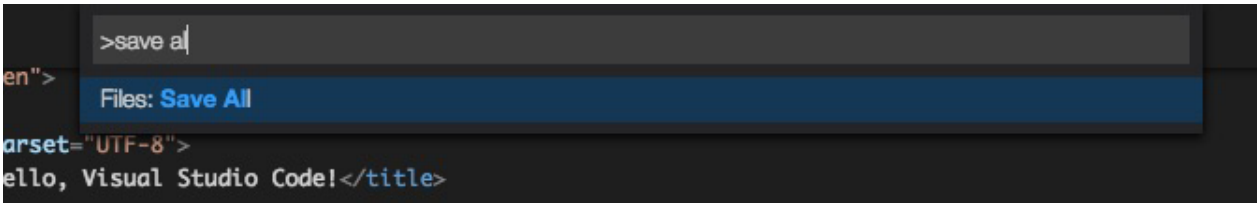
```
/// <reference path="typings/jquery/jquery.d.ts"/>
$('#rotate').on('click', function () {
    $('#rotate').animate({ zIndex : 1 },{
        duration : 1000,
        step : function (value) {
            $(this).css({
                transform : 'rotate(' + (value * 360) + 'deg)'
            });
        },
        complete : function () {
            $('#rotate').css('zIndex', 0);
        }
    });
});
```

2.2.6. ファイルを保存する / オートセーブ機能を利用する

作業中のファイルで、未保存の変更があるファイルには、〇マークが表示されています。実際の作業中には、複数のファイルをまたがった IntelliSense による補完やデータ損失の防止などの理由から、頻繁に保存することをお勧めします。

現在エディタで開いている 1 つのファイルを保存するには、⌘S キー（Windows では Ctrl+S キー）を押すか、[File]メニューから[Save]をクリックします。

またすべての作業中のファイルを保存するには、同様に[File]メニューから[Save All]をクリックするか、コマンドパレットから[Files: Save All]を選択します。



さらに、Visual Studio Code には、作業ディレクトリのすべてのファイルに対して、行った編集を即時保存する、オートセーブ機能があります。この機能を有効にするには[File]メニューから[Enable Auto Save]を選択するか、コマンドパレットから同様の項目を選択することで、すぐに利用することができます。無効にする方法も同様です。

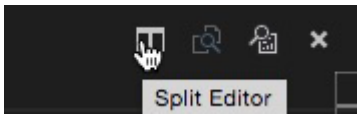
2.2.7. Side by Side の編集機能を利用する

では最後に README を追加しましょう。
作業ディレクトリに README.md を追加します。（2.2.3 章）

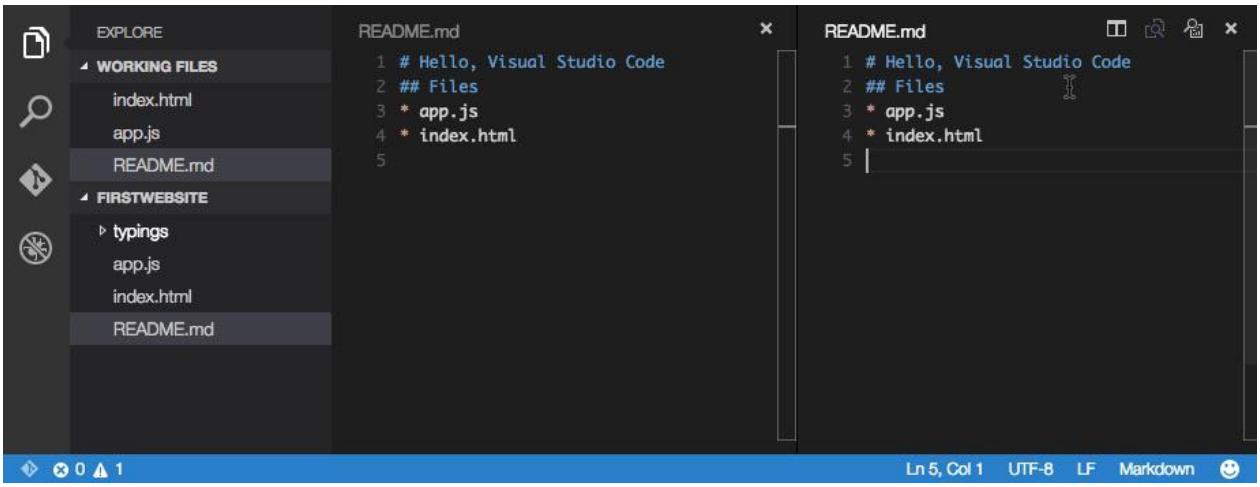
続いて以下の行を入力します。

```
1 # Hello, Visual Studio Code
2 ## Files
3 * app.js
4 * index.html
```

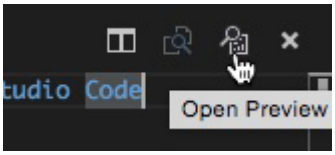
次にエディタ左上にある、[Split Editor]ボタンをクリックします。



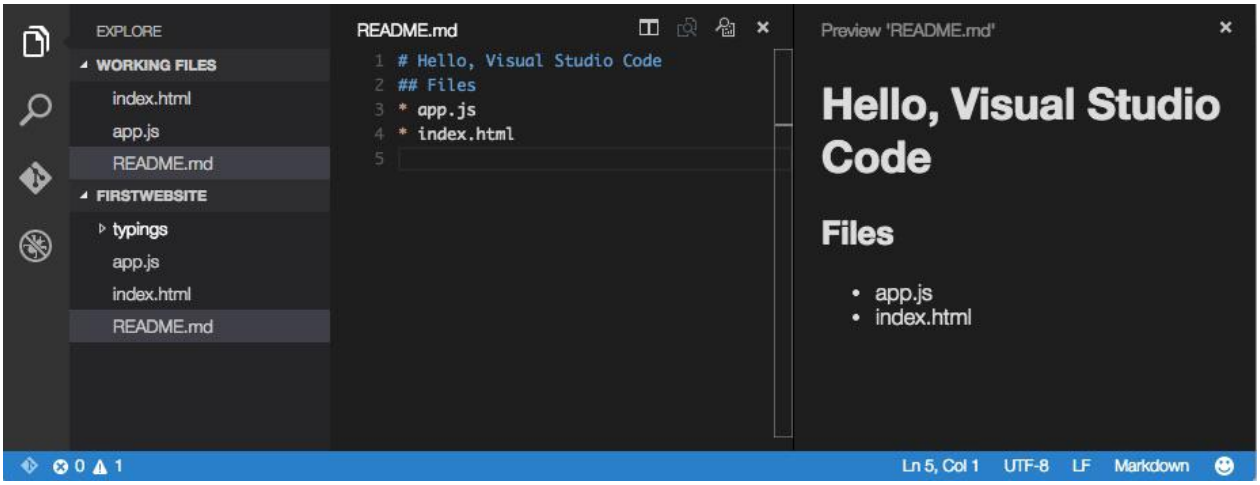
これにより、エディタ領域が 2 つに分割されます。この分割は最大 3 つまで行うことができます。



続いて、右側エディタの右上にある[Open Preview]ボタンをクリックします。



この操作により、右側パネルが Markdown のプレビュー画面となります。

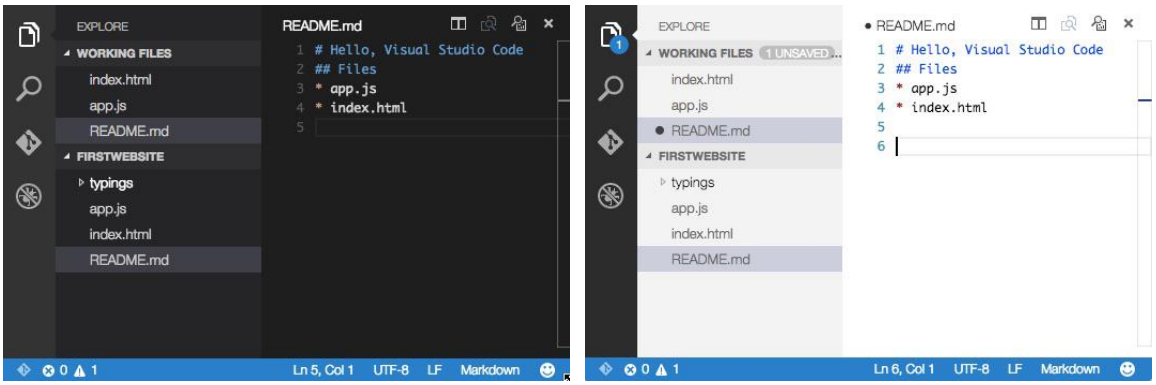


また左側を編集すると、保存・未保存に関わらず、右側パネルがリアルタイムに更新されます。このように Visual Studio Code では、パネルを分割して別々のファイルを変更することや、あるファイルに対して同時にプレビューを表示することも可能です。

2.3. Visual Studio Code のカスタマイズ

2.3.1. テーマの変更

Visual Studio Code にはデフォルトで Light Theme と Dark Theme が搭載されています。テーマの変更は[View]メニューの[Theme]から選択するか、コマンドパレットで[View: Change to Dark/Light Theme]を選択してください。



（左：Dark Theme　右：Light Theme）

2.3.2. エディタの変更

Visual Studio Code のエディタ部分の表示設定も簡単にカスタマイズすることができます。
詳細に関しましては、 Customize Visual Studio Code にある Settings の項目 を参照してください。

2.3.3. キーバインディングの変更

Visual Studio Code のショートカットやキーバインディングをカスタマイズすることも可能です。
詳細に関しましては、 Customize Visual Studio Code にある Customizing Keyboard Shortcuts の項目を参照してください。

2.3.4. ユーザースニペットの追加

Visual Studio Code では Emmet の他に、独自のスニペットを定義（**ユーザースニペット**）して使うことができます。ユーザースニペットの定義は[File]メニューの[Preference]メニューから[User Snippets]を選択することで、カスタマイズすることが可能です。詳細に関しましては、Editing Evolved にある Snippets の項目を参照してください。

3. Visual Studio Code を用いた Node.js アプリケーションの開発

この章では npm の概要と環境構築、および Visual Studio Code を利用した Node.js アプリケーションの開発についてご紹介します。

3.1. Node.js とは

Node.js とはサーバーで動作する JavaScript の実行環境です。Node.js の主な特徴としては、接続ごとにプロセス生成やメモリの割り当てを行わず、Node.js のプロセス内のイベントを発生させて処理を行うことで、多数の同時接続を処理することが可能となる点が挙げられます。また内部としてノンブロッキング I/O などを採用し、軽量で効率の良い動作を行うフレームワークを実現しています。Visual Studio Code も内部的に Node.js を用いて動作しています。

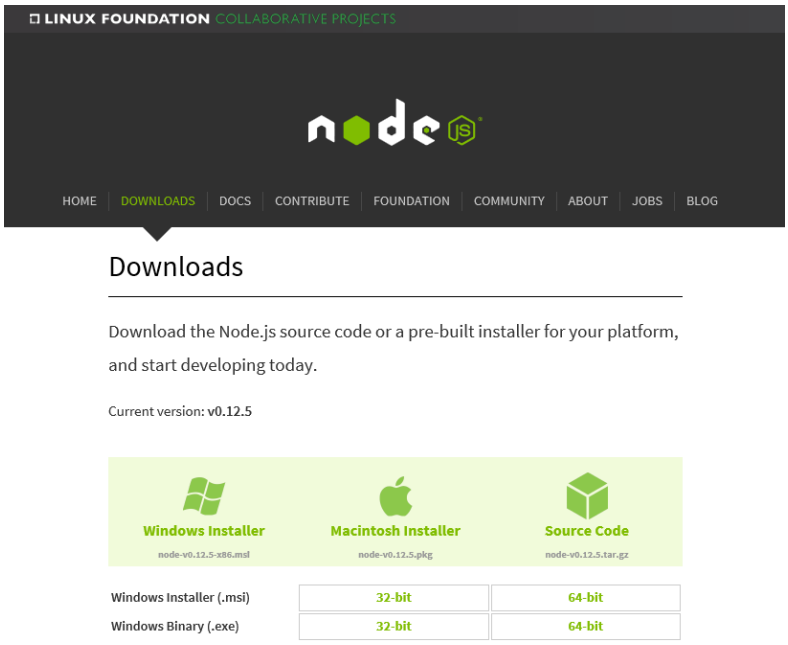
Node.js には多くのモジュールが公開されており、このモジュールを組み合わせることで非常に様々なことが実現可能となります。このモジュールを管理するツールとして「**npm**」（Node Package Manager）があります。

また **gulp** は Node.js 上で動作する、タスクを自動化するためのビルドツールです。例えばソースコードの圧縮（minify）や TypeScript のコンパイルなどをタスクとして定義しておくことで、Visual Studio Code 上から簡単に実行させることができます。

今回は Node.js 上で動作する MVC フレームワークである Express と、この Express を利用した Web アプリケーションのひな形を作ることが出来るパッケージである express-generator を用いて、簡単な Node.js アプリケーションを制作する方法をご紹介します。

3.2. Node.js アプリケーション開発に向けた環境構築

この章では、アプリケーションの開発に必要な Node.js・npm・express、また Debug に必要な Mono のインストール方法をご紹介します。

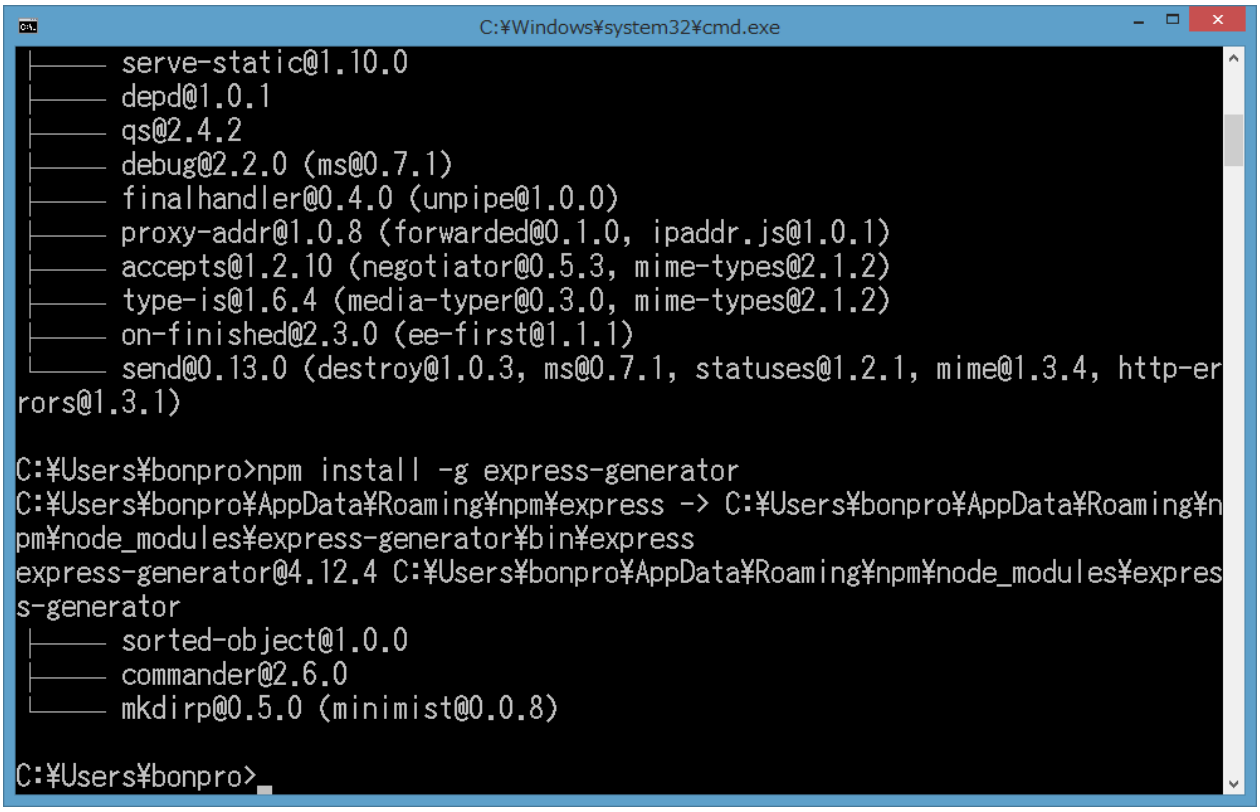


3.2.1. Windows での環境構築

1. https://nodejs.org/download/ にアクセスします。
2. Windows Installer をクリックします。（2015 年 7 月現在、node-v0.12.5-x86.msi がダウンロードされます）
3. インストーラの手順に従って使用許諾契約の同意とインストールを行います。

以上で Node.js のインストールが完了しました。続いて Express のインストールを行います。

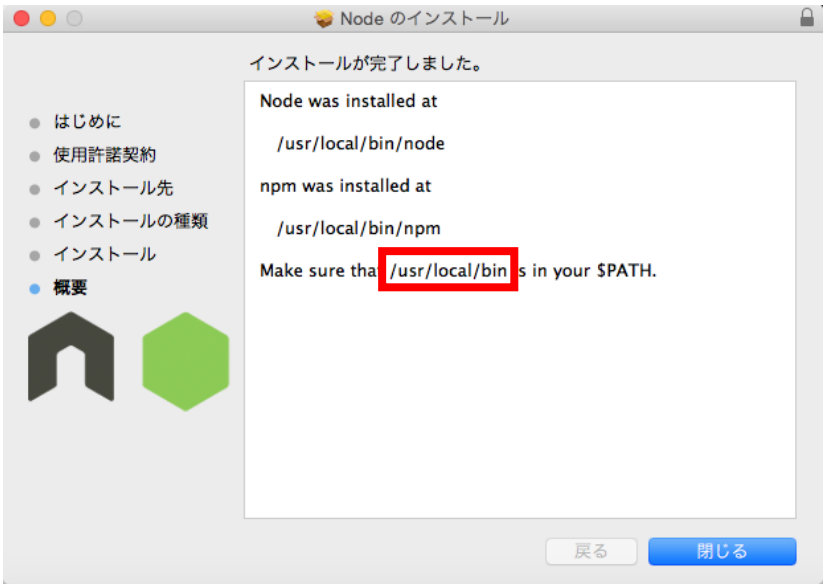
4. コマンドプロンプトを開きます。
5. npm install –g express express-generator を行います。



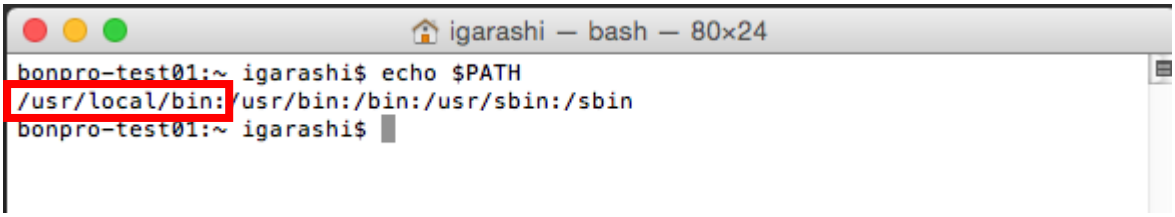
以上で Express のインストールは完了です。
Windows の場合、デバッグ用の Mono のインストールは不要ですので、以上ですべてのインストールが完了しました。

3.2.2. Mac での環境構築

1. https://nodejs.org/download/ にアクセスします。
2. Macintosh Installer をクリックします。（2015 年 7 月現在、node-v0.12.5.pkg がダウンロードされます）
3. ダウンロードされたファイルを開くと、インストーラが起動します。
4. インストーラの手順に従って使用許諾契約の同意とインストールを行います。
5. インストーラが完了すると、以下の画面が表示されます。“Make sure that **** in your $PATH.”（****は任意の文字列）の 1 行があることを確認し、****のパスをメモします。



6. Terminal を開きます。
7. echo $PATH を入力します。
8. 結果に 4.で確認したパスが含まれているかを確認します。



9. 含まれていない場合は、シェルの設定ファイルなどを用いて環境変数にパスを追加します。

以上で Node.js のインストールが完了しました。続いて Express のインストールを行います。

10. Terminal を開きます。
11. sudo npm install –g express express-generator を実行します。

以上で Express のインストールは完了です。最後にデバッグ用の Mono のインストールを行います。

12. http://www.mono-project.com/download/ にアクセスします。
13. Mac OS X の項目にある[Download Mono MDK]ボタンをクリックします。(2015 年 7 月現在、MonoFramework-MDK-4.0.2.5.macos10.xamarin.x86.pkg がダウンロードされます)
14. ダウンロードされたファイルを開くと、インストーラが起動します。
15. インストーラの手順に従って使用許諾契約の同意とインストールを行います。

以上ですべてのインストールが完了しました。

3.2.3. Linux での環境構築

3.2.2 章の手順と同様にしてインストールを行うことができます。適宜お使いの環境に合わせて読み替えてください。また apt-get や、epel リポジトリが追加された yum など、パッケージマネージャーを利用してインストールすることも可能です。

ただしデバッグ時に必要となる **Mono のバージョン**は **3.12 以降**でなければならないことにご注意ください。

3.3. Node.js アプリケーションの構築

では、実際に Node.js アプリケーションを構築する方法をご紹介します。
まずは express-generator を用いて、Express フレームワークを使用したアプリケーションのひな形を作成します。

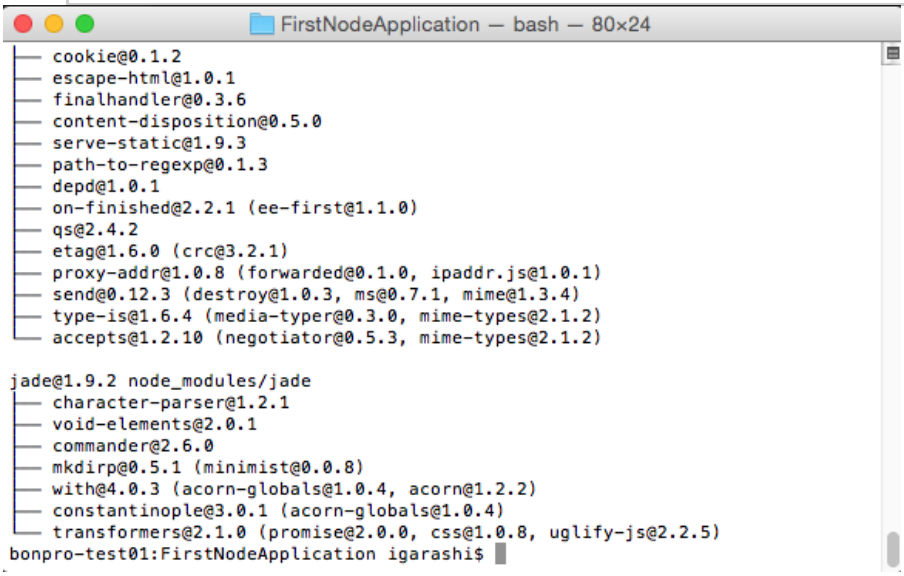
ターミナル、もしくはコマンドプロンプトを起動し、適当なフォルダで以下のコマンドを実行します。

```
express FirstNodeApplication
```

```
   create : FirstNodeApplication/package.json
   create : FirstNodeApplication/app.js
   create : FirstNodeApplication/public
   create : FirstNodeApplication/public/javascripts
   create : FirstNodeApplication/public/images
   create : FirstNodeApplication/public/stylesheets
   create : FirstNodeApplication/public/stylesheets/style.css
   create : FirstNodeApplication/routes
   create : FirstNodeApplication/routes/index.js
   create : FirstNodeApplication/routes/users.js
   create : FirstNodeApplication/views
   create : FirstNodeApplication/views/index.jade
   create : FirstNodeApplication/views/layout.jade
   create : FirstNodeApplication/views/error.jade
   create : FirstNodeApplication/bin
   create : FirstNodeApplication/bin/www

   install dependencies:
     $ cd FirstNodeApplication && npm install

   run the app:
     $ DEBUG=FirstNodeApplication:* npm start

bonpro-test01:~ igarashi$
```

コマンド実行後、FirstNodeApplication フォルダが自動的に作成され、ひな形が作成されていることが確認できます。続いて以下のコマンドを入力します。

```
cd FirstNodeApplication
npm install
```



このコマンドでは、作成された FirstNodeApplication に移動し、npm install を実行することで、package.json に書かれたモジュールを、npm を用いてインストールしています。

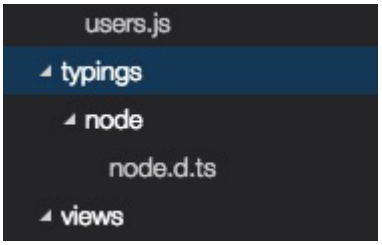
コマンド実行後、code . を実行する（ただし Mac の場合は 2.1.4 章を設定している必要があります）か、Visual Studio Code を起動し、作成したディレクトリを開きます。

続いて、参照する Node.js のモジュールを追加する方法をご紹介します。

package.json を開きます。
例として jQuery を追加するには、"dependencies"以下に jquery を追加します。この時、Visual Studio Code は npm からパッケージ一覧情報の取得、パッケージバージョン情報の取得を行い、IntelliSense を通して入力支援を行います。ユーザーは、パッケージ名やバージョンの確認をエディタのみで完結させることができ（例：下画像）、package.json の記述に専念することができます。

次に app.js を開きます。14 行目の__dirname が緑色になっていることが確認できます。カーソルをこの部分に合わせると、以下のような電球マークが表示されます。

電球マークをクリックし、[Add /// reference to 'node/node.d.ts']を選択します。
これは Express が生成した app.js には TypeScript の型定義ファイルおよびその参照が入っていないため、Visual Studio Code 側が正常に補完できないことを示しており、この操作により自動で型定義ファイルがダウンロードされ IntelliSense が強化されます。エクスプローラには typings ディレクトリが自動で作成されており、Visual Studio Code によって型定義ファイルが自動でダウンロードされたことが確認できます。



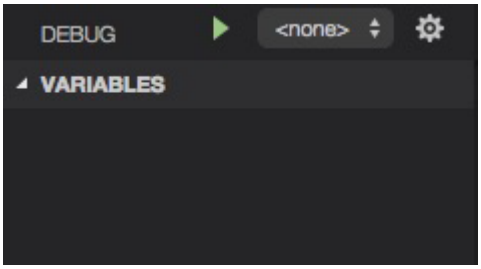
最後に app.js の 59 行目（module.exports の前の行）に以下のコードを追加します。

```
app.listen(5000);
```

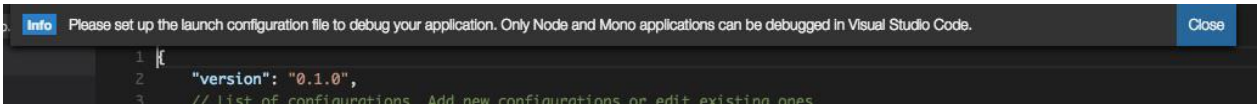
3.4. Node.js アプリケーションのデバッグ

では作成したアプリケーションの実行方法をご紹介します。
ビューバーからデバッグボタンを選択します。

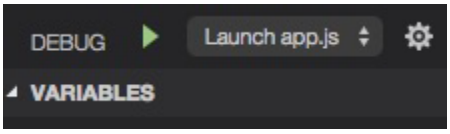
ウインドウ左上に着目してください。



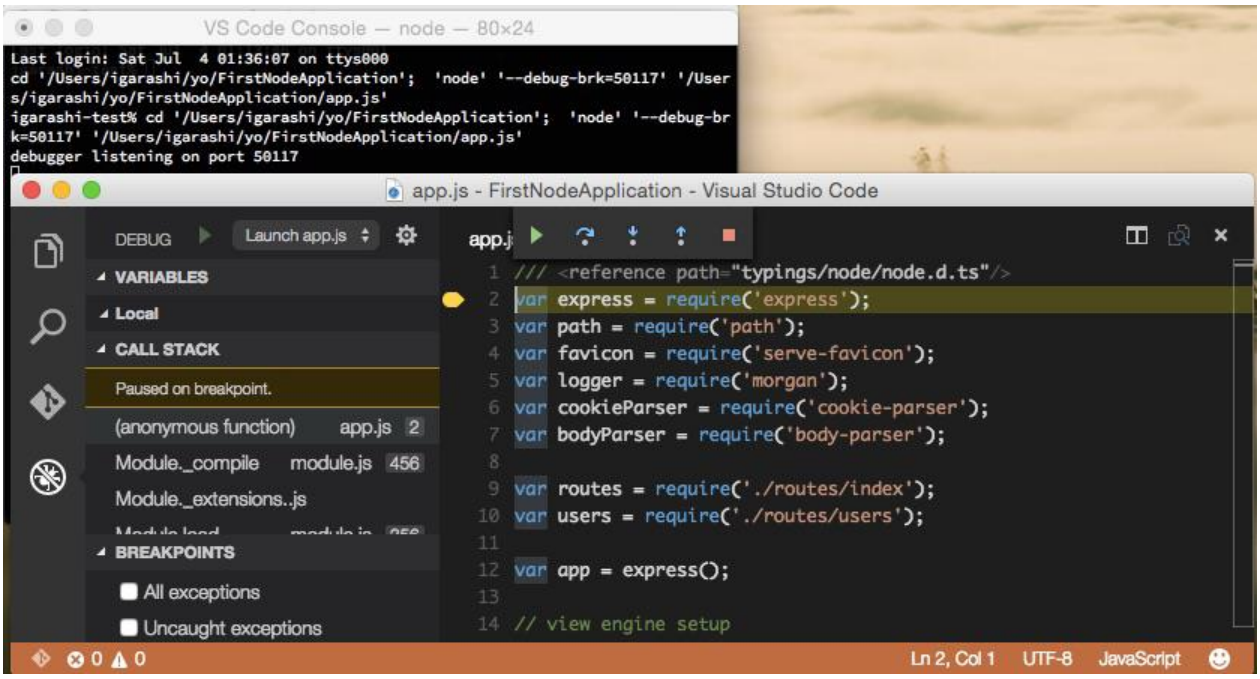
現状は何もデバッグの構成ファイルがないため<none>となっています。
では、再生ボタンを押してみましょう。



すると自動で設定ファイル（.settings/launch.json に格納されます）が作成されました。今回は app.js をそのまま起動しますので、設定ファイルを変更する必要はありません。再度左上に着目してください。

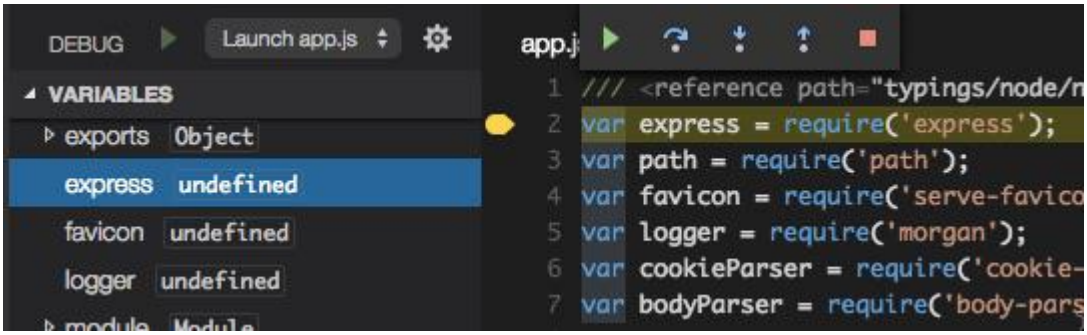


今度は設定ファイルにより、[Launch app.js]（設定ファイルの name 値）が追加されました。再度再生ボタンをクリックします。



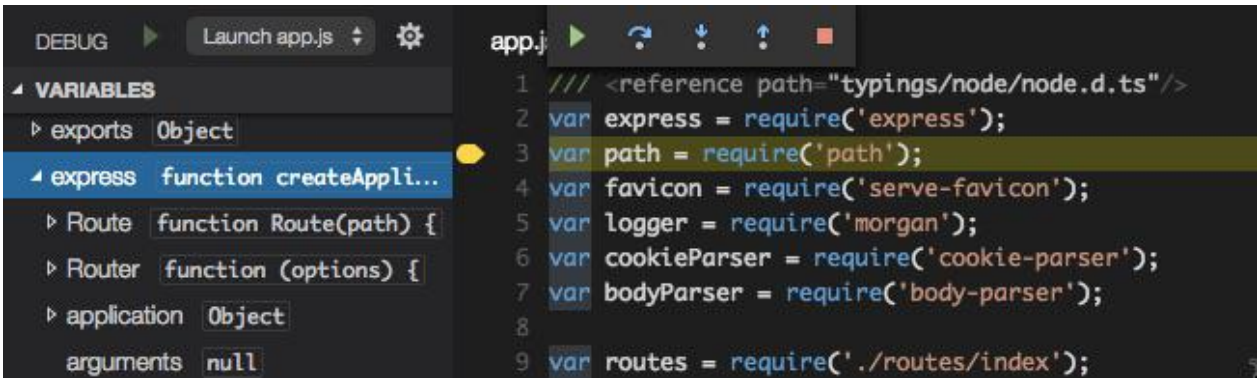
先ほどとは異なり、今回は[Launch app.js]に記述された設定に基づいてデバッグが実行されます。デバッグ実行時にはステータスバーの色がオレンジ色となります。また、デフォルトでは stopOnEntry 値が true となっているため、デバッグ開始時にブレークポイントがかかるようになっています。

画面左側の VARIABLES（変数）グループを確認します。

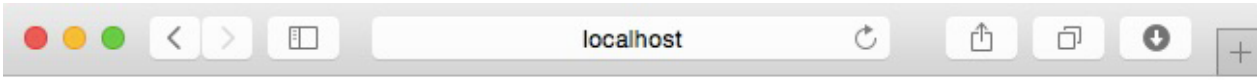


上図の通り、現時点では express の値が undefined（未定義）となっていることがわかります。
では[Step Over]ボタン、もしくは F10 キーを押して、次の命令に進めましょう。

ここで再度変数グループを確認すると、1 行目が実行されたことにより、express の値が変化していることがわかります。



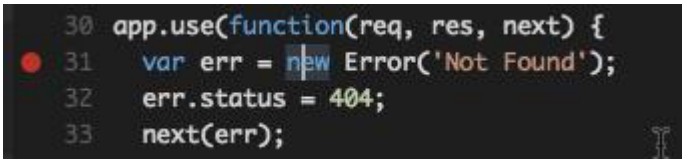
では再生ボタンをクリックして、動作を再開させましょう。
ブラウザを開き、localhost:5000 にアクセスします。
以下のような Web サイトが表示されれば、アプリケーションが正常に動作しています。



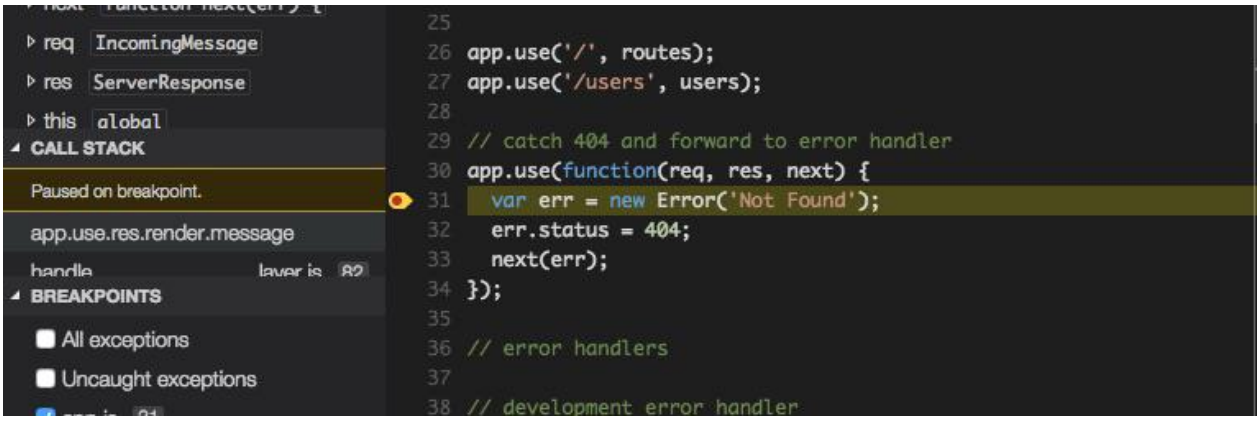
Express

Welcome to Express

では Visual Studio Code に戻り、app.js の 31 行目の行番号左側をクリックし、赤色の丸印を付けます。これはブレークポイントが設定されたことを表し、処理がこのコードに到達すると、自動的にデバッガが表示されることを意味します。app.js の 31 行目はコンテンツが存在しない時に 404 エラーを返す処理を定義しています。



再びブラウザに戻り、localhost:5000/foobar などの定義されていないアドレスにアクセスします。この動作を行った直後に Visual Studio Code がアクティブになり、先ほど設定した箇所で動作が停止していることが確認できます。



再生ボタンをクリックすることで、処理を続行できます。

4. Visual Studio Code を用いた ASP.NET 5 アプリケーションの開発

この章では ASP.NET 5 の概要と環境構築、および Visual Studio Code を利用した ASP.NET 5 アプリケーションの開発についてご紹介します。

4.1. ASP.NET 5 とは

ASP.NET は.NET Framework をベースにした Web アプリケーション開発フレームワークです。基本的な Web アプリケーション制御や Twitter 認証を用いた外部認証などのユーザー管理などにもデフォルトで対応しており、高機能な Web アプリケーションを比較的容易に制作することが可能となります。

ASP.NET アプリケーションは動的に Web ページを生成するため、Web アプリケーションをホストするランタイムが必要です。従来の ASP.NET は、依存関係などの問題から Windows プラットフォーム上でしか動作することができませんが、ASP.NET 5 からはクロスプラットフォームをサポート、そしてランタイムがオープンソース化されました。

さらに npm や Grunt/gulp をはじめ、現在主流の各種ユーティリティを取り入れた開発を標準サポートし、従来の Web 開発により近い感覚で開発を行うことができます。

今回は ASP.NET 5 MVC と、そのひな形を作ることが出来るパッケージである Yeoman を用いて、簡単な ASP.NET 5 MVC アプリケーションを制作する方法をご紹介します。

なお、今回作成するアプリケーションのプロジェクトは以下のリンクからダウンロードすることもできます。
https://github.com/bonprosoft/FirstASPNet5Application

4.2. ASP.NET 5 アプリケーションの構成

従来の ASP.NET アプリケーションのホストには IIS を用いておりましたが、IIS には Windows ライブラリと強い依存関係があり、Linux や Mac では正常に動作させることができません。そこで ASP.NET 5 以降は、ランタイムとして DNX（.NET Execution Environment）を用いてホストすることが可能となっています。現状 Mac や Linux でのホストには、この DNX を用いる必要があります。

DNX には以下に主要な 3 つのコマンドがあります。

- dnvm ： DNX ランタイムの管理を行います
- dnx ： アプリケーションの実行などを制御します

・ dnu ： パッケージの管理を行います（NuGet）

また開発時には、ひな形生成（スキャフォールディング）を行う Yeoman や、タスクランナーである Grunt などの各種ツールも組み合わせることで、より効率的にアプリケーションの開発を行うことができます。

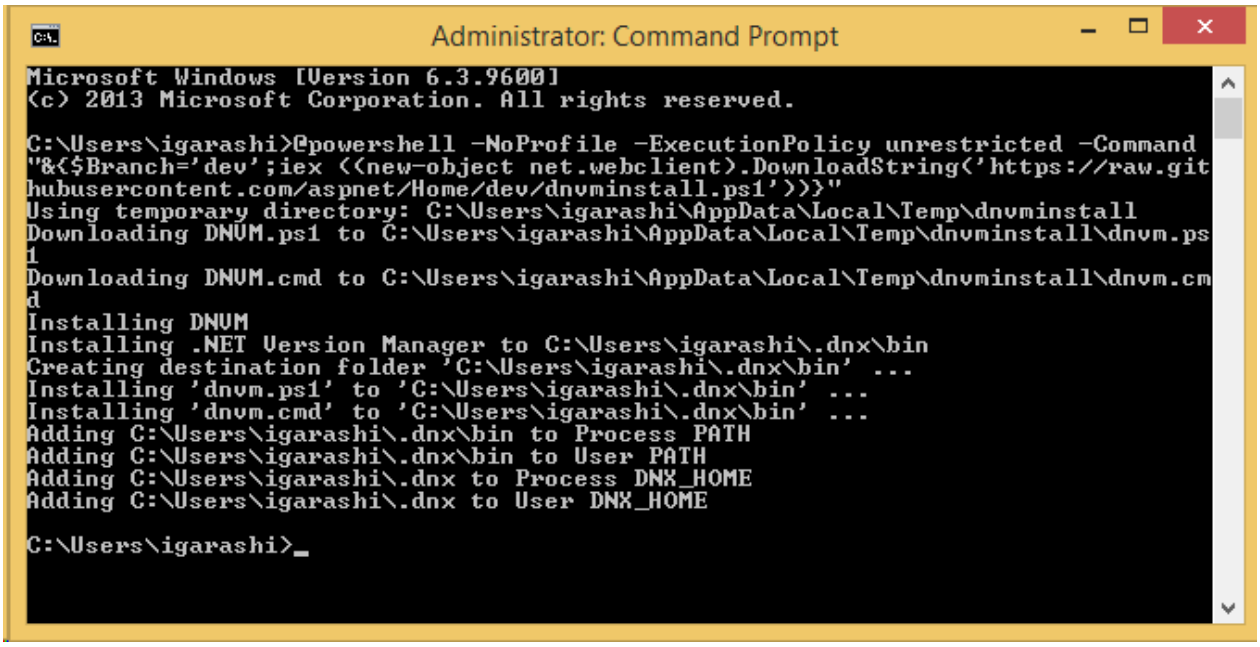
また ASP.NET 5 アプリケーションのファイル構造も、以前のものと比べて大きく変化しました。ASP.NET 5 プロジェクトにおいて静的コンテンツは *wwwroot* フォルダ内に配置され、プロジェクトの設定ファイルも JSON 形式で管理（*project.json*）されます。さらに 3.3 章でご紹介した package.json（npm の設定ファイル）と同様、Visual Studio Code は標準で project.json の補完に対応しています。そのため、ユーザーは NuGet を意識することなく、従来の Web 開発とほぼ同様の手法でプロジェクトの管理を行うことができます。

4.3. ASP.NET 5 アプリケーション開発に向けた環境構築

4.3.1. Windows での環境構築

Visual Studio 2015 が導入されている場合には、既に ASP.NET 5 のアプリケーション開発環境は構築されていますので、Yeoman などの各種ツールの導入へスキップしてください。
もし Visual Studio 2015 がインストールされていない Windows 環境を用いる場合には、コマンドプロンプトを開いて以下のコマンドを実行します。

```
@powershell -NoProfile -ExecutionPolicy unrestricted -Command "&{$Branch='dev';iex ((new-object net.webclient).DownloadString('https://raw.githubusercontent.com/aspnet/Home/dev/dnvminstall.ps1'))}"
```



インストールが終了したら、一度コマンドプロンプトを終了し、再度コマンドプロンプトを起動して以下のコマンドを実行します。

```
where dnvm
```

このコマンドの結果、何かパスが返されれば ASP.NET 5 の実行環境のインストールが完了しました。続いて yeoman などの各種ツールの導入を行います。コマンドプロンプトを用いて、以下のコマンドを実行します。（npm の実行には、3.2.1 章でご紹介した Node.js のインストールが必要です）

```
npm install -g yo grunt-cli generator-aspnet bower
```

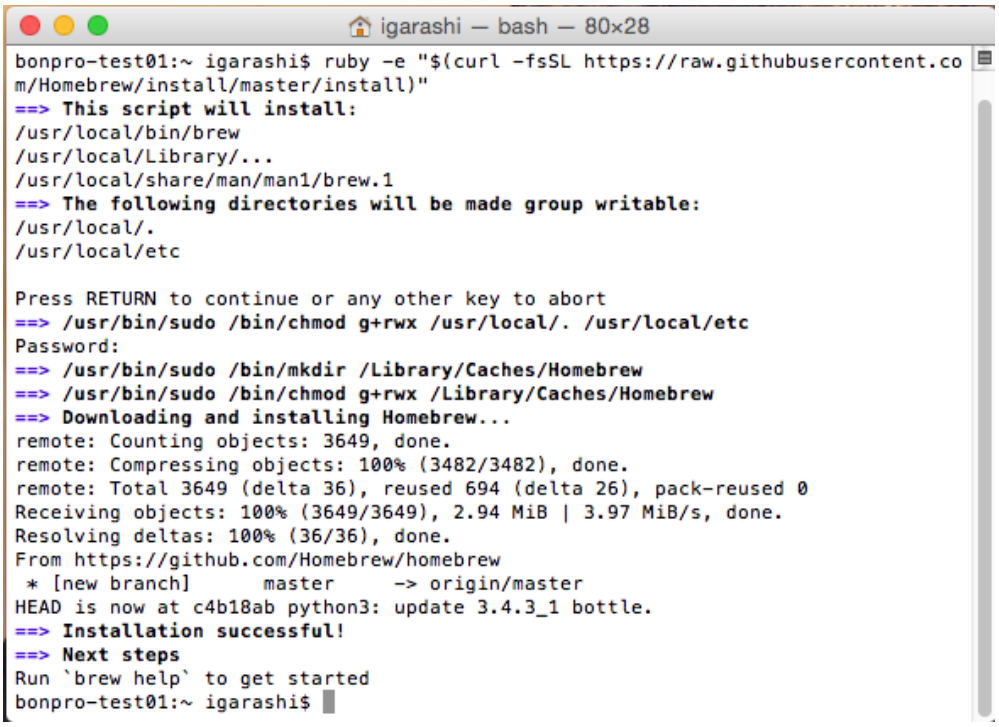
以上ですべてのインストールが完了しました。

4.3.2. Mac での環境構築

現状、Mac で DNVM をインストールする最も簡単な方法は、Homebrew を用いてインストールする方法です。ここでは Homebrew のインストール方法、Homebrew を用いた DNVM のインストール方法をご紹介します。

1. 以下のコマンドを実行します。

```
ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"
```



途中パスワードが求められた場合には、ログインしているユーザーのパスワードを入力してください。以上で Homebrew のインストールが完了しました。以降 brew コマンドを用いて DNVM のインストールを行います。

2. 以下のコマンドを実行します。

```
brew tap aspnet/dnx
brew update
brew install dnvm
```

3. 最後にお使いのシェルの設定ファイルに以下のコードを追加します。（通常は.bash_profile に記述してください）

```
source dnvm.sh
```

以上で DNVM のインストールが完了しました。最後に、Yeoman などの各種ツールの導入を行います。以下のコマンドを実行します。（npm の実行には、3.2.2 章でご紹介した npm のインストールが必要です）

```
sudo npm install -g yo grunt-cli generator-aspnet bower
```

以上ですべてのインストールが完了しました。

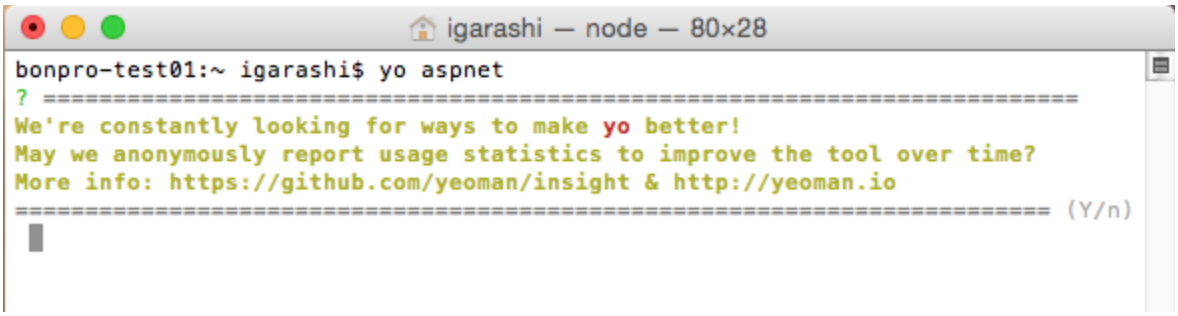
4.3.3. Linux での環境構築

お使いの環境によって異なります。詳細は Getting Started with ASP.NET 5 and DNX にある Linux の項目を参照してください。

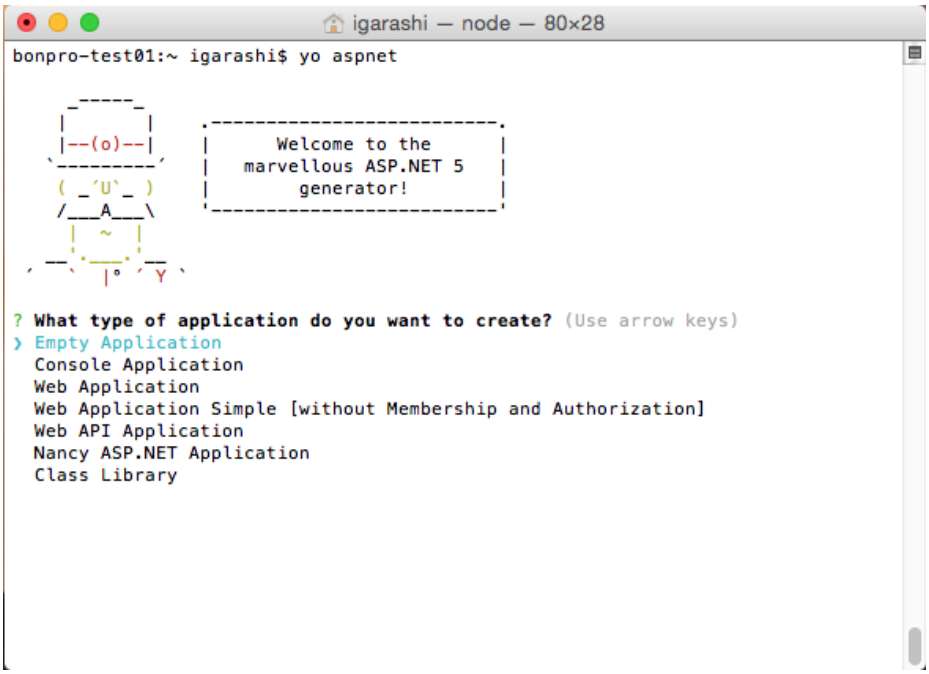
4.4. ASP.NET 5 アプリケーションの構築

まず初めに、Yeoman を用いて ASP.NET 5 アプリケーションのテンプレートを作成します。
適当なディレクトリに移動し、以下のコマンドを実行します。

```
1 yo aspnet
```



初回起動時には匿名レポートの可否について聞かれる場合がありますので、Yes か No を入力してください。



Yeoman が起動すると、どのアプリケーションテンプレートを利用するかを選択する画面が表示されます。今回は Web Application を作成しますので、上下の矢印キーを用いて Web Application を選択し、Enter キーを押してください。

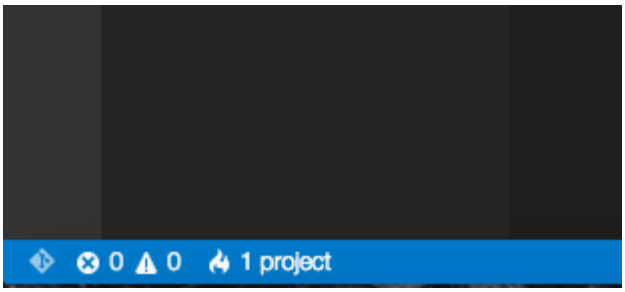
次の画面でプロジェクト名を聞かれますので、そのまま Enter キーを押してください。(この場合、デフォルトの"WebApplication"がプロジェクト名となります。変更するには任意のアプリケーション名を入力してください。その場合、WebApplication を入力した名前として読み替えてください)

以上で入力した名前のディレクトリが作成され、基本的な Web アプリケーションのテンプレートが生成されました。
続いて作成されたディレクトリに対して cd コマンドを用いて移動し、以下のコマンドを実行します。

```
1 dnu restore
```

この操作により project.json に書かれた依存モジュールを元に、NuGet から自動的にパッケージを導入することができます。この操作は Visual Studio Code からも行うことができます。

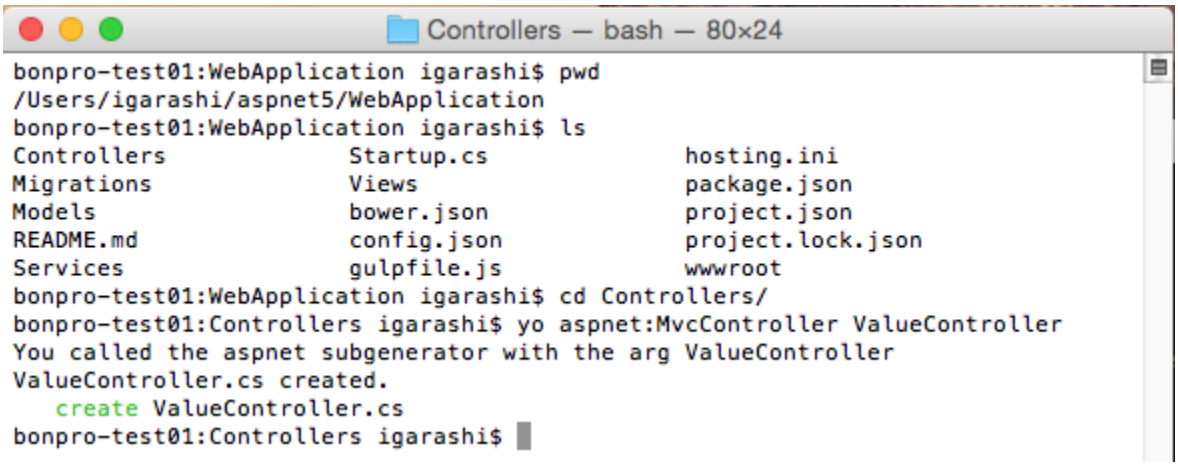
Visual Studio Code を起動し、作成したディレクトリを開きます。



画面下のステータスバーに 1 project と表示され、Visual Studio Code にプロジェクトとして正常に認識されているかどうかを確認します。ここで一旦 4.5 章を参考に、ASP.NET アプリケーションを実行してみましょう。

では、続いて Yeoman の Sub Generator 機能を用いて、コントローラと View を追加しましょう。
ターミナルに戻り、作成したディレクトリ（今回は WebApplication）で、以下のコマンドを実行します。

```
cd Controllers
yo aspnet:MvcController ValueController
```



以上で Controller フォルダ内に ValueController.cs が作成されました。続いて、このコントローラの Index アクションに対応する View の作成を行いましょう。

```
cd ../Views/
mkdir Value
yo aspnet:MvcView Index
```

以上で Value コントローラの Index アクションに対応する View が作成されました。
では再び Visual Studio Code に戻りましょう。

エクスプローラから Controllers フォルダの ValueController.cs を開きます。

9 行目の namespace を以下のように変更します。

```
9  namespace WebApplication.Controllers
```

さらに Index アクションの中身を以下のように変更します。

```
14 public IActionResult Index(string value)
15 {
16     ViewBag.Value = value;
17     return View();
18 }
```

以上で ViewBag の Value フィールドにクエリとして与えられた value の値を格納する Index アクションを作成しました。
では次に View の編集を行います。
エクスプローラから Views フォルダの中の Values フォルダから Index.cshtml を開きます。

7 行目（ファイル末尾）以降を以下のように変更します。

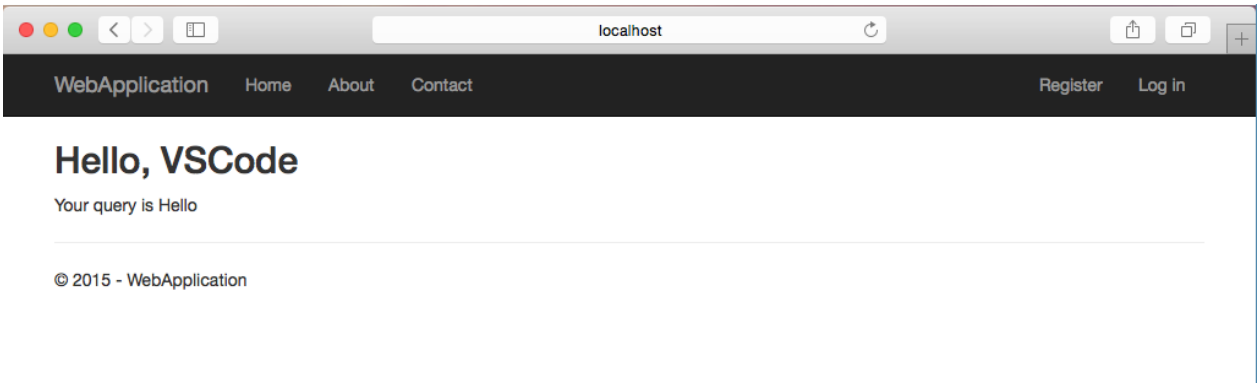
```
14 <h2>Hello, VSCode</h2>
15 <p>Your query is @ViewBag.Value</p>
```

以上で ViewBag の Value フィールドに格納された値を表示する View を作成しました。
4.5 章を参考に、作成したアプリケーションを実行してみましょう。

Web ブラウザで、例として以下の URL にアクセスします。（ポート番号 5000 は 4.5 章の hosting.ini で指定されている値に合わせて変更してください。）

http://localhost:5000/Value/Index?value=Hello

この時クエリとして与えた文字列が表示されれば正常に動作しています。



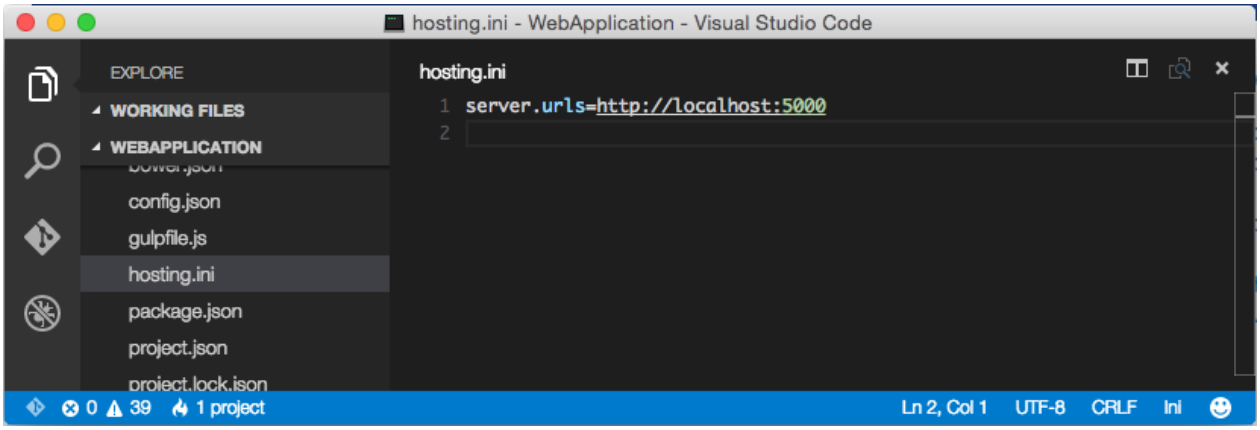
Yeoman の ASP.NET Generator には、これ以外にも様々な Sub Generator が搭載されています。詳細に関しましては、yo aspnet --help コマンドを実行するか、以下の URL を参照してください。

https://github.com/OmniSharp/generator-aspnet

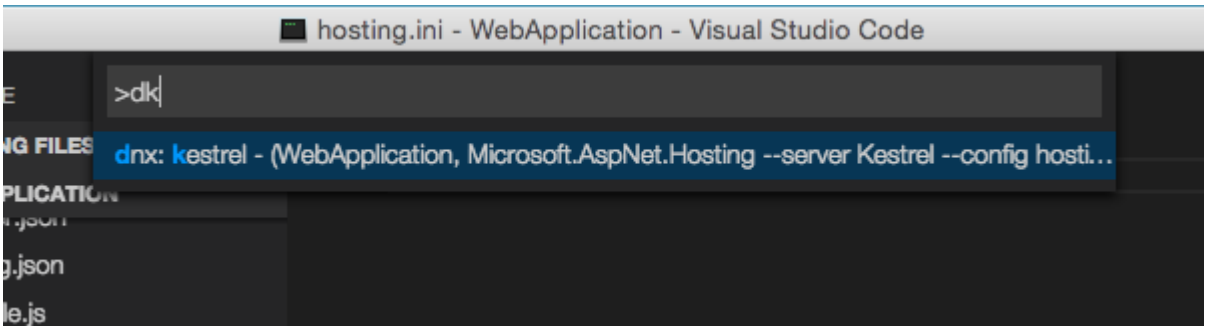
4.5. ASP.NET 5 アプリケーションの実行とデバッグ

はじめに作成した ASP.NET5 アプリケーションの実行方法をご紹介します。

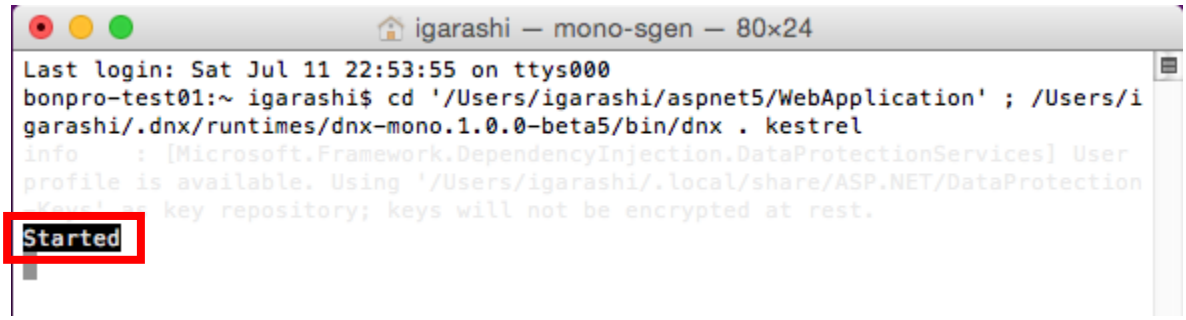
まずエクスプローラから hosting.ini を開いてください。通常はこのファイル内の server.urls で指定される URL でホストされます。ここではポート番号が 5000 番であることが分かります。この番号は動作させた Web アプリケーションを表示する際に必要となりますので、覚えておきましょう。



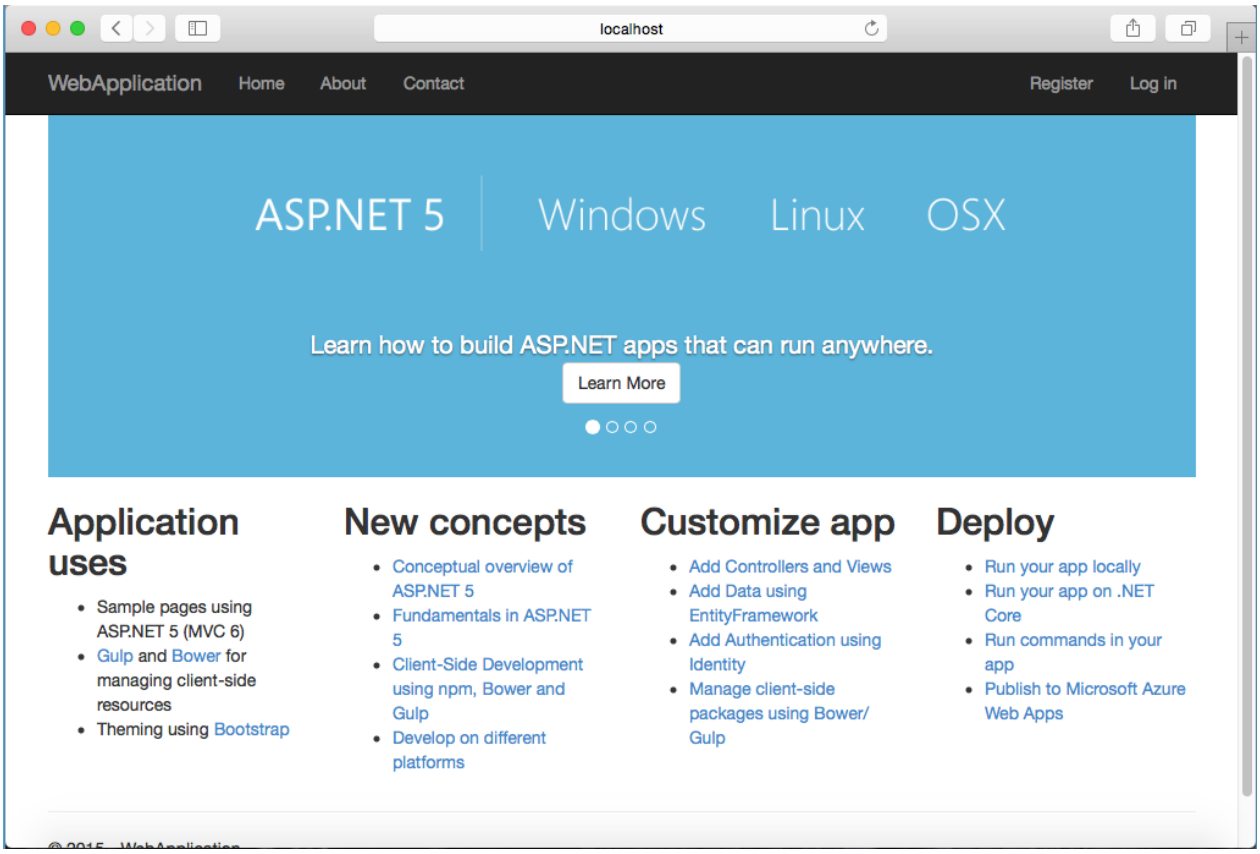
続いて⇧⌘P（Windows は Shift+Ctrl+P）キーを同時に押してコマンドパレットを表示し、[dnx: kestrel]を選択します。



これを実行すると、以下のような画面が表示され、[Started]と表示されれば正常にアプリケーションが動作しています。



お使いの Web ブラウザを用いて http://localhost:5000 （先ほど確認した番号が 5000 であった場合）にアクセスします。



正常に動作している場合、このような Welcome スクリーンが表示されます。

続いて ASP.NET5 アプリケーションのデバッグについてご紹介します。
2015 年 7 月現在（VS Code v0.5.0）、ASP.NET5 アプリケーションのデバッグには対応しておりません。しかしながら Visual Studio Code はほぼ毎月アップデートが提供されており、チームとしても将来的には対応する方向で開発が進んでいます。[1] Visual Studio Code のアップデート内容は以下のリンクからご確認いただけます。

https://code.visualstudio.com/updates

[1] https://code.visualstudio.com/Docs/ASPnet5

5. Visual Studio Code を用いた TypeScript アプリケーションの開発

5.1. TypeScript とは

TypeScript は JavaScript を拡張し、静的型付けを含む様々な言語機能をもった言語です。記述した TypeScript コードは標準的な JavaScript に変換でき、一般的な Web サーバーでホストすることができます。また TypeScript 中に JavaScript をそのまま記述することができ、ユーザーの好みに合わせて部分的に TypeScript の言語機能を用いることもできます。

TypeScript には前述の静的型付けをはじめ、クラス、インターフェイス、ジェネリックなど、様々な優れた言語仕様があります。また TypeScript 1.6 からは async/await をはじめ、数々の ECMAScript 6 の新機能が取り入れられています。さらに静的型付けの恩恵として、各種エディタの補完機能がより強化され、リアルタイムに型エラーを検出することができます。また DefinitelyTyped チームによって、既存のライブラリの型情報を捕捉する型定義ファイルが作成・公開されており、jQuery/AngularJS/D3.js をはじめ、最新の JavaScript ライブラリに広く対応しています。TypeScript や DefinitelyTyped は GitHub でオープンソースとして公開されています。

TypeScript のロードマップは以下のリンクから参照してください。

https://github.com/Microsoft/TypeScript/wiki/Roadmap

なお、今回作成するアプリケーションのプロジェクトは以下のリンクからダウンロードすることもできます。
https://github.com/bonprosoft/FirstTypeScriptApplication

5.2. TypeScript アプリケーション開発に向けた環境構築

5.2.1. Windows での環境構築

Visual Studio 2013 および 2015 が導入されている場合には、TypeScript for Visual Studio 内の tsc（TypeScript Compiler）が利用できます。2013 の方はこちら、2015 の方はこちらのリンクからインストールしてください。

もし Visual Studio 2015 がインストールされていない Windows 環境を用いる場合には、コマンドプロンプトを開いて以下のコマンドを実行します。（npm の実行には、3.2.1 章でご紹介した npm のインストールが必要です）

```
1 sudo npm install -g typescript
```

続いて型定義ファイル用のパッケージマネージャーである、tsd をダウンロードします。

```
sudo npm install -g tsd
```

以上ですべてのインストールが完了しました。

5.2.2. Mac での環境構築

現状、TypeScript をインストールする最も簡単な方法は npm を用いる方法です。
ここでは npm を用いた方法をご紹介します。（npm の実行には、3.2.2 章でご紹介した npm のインストールが必要です）

```
sudo npm install -g typescript
```

続いて型定義ファイル用のパッケージマネージャーである、tsd をダウンロードします。

```
sudo npm install -g tsd
```

以上ですべてのインストールが完了しました。

5.2.3. Linux での環境構築

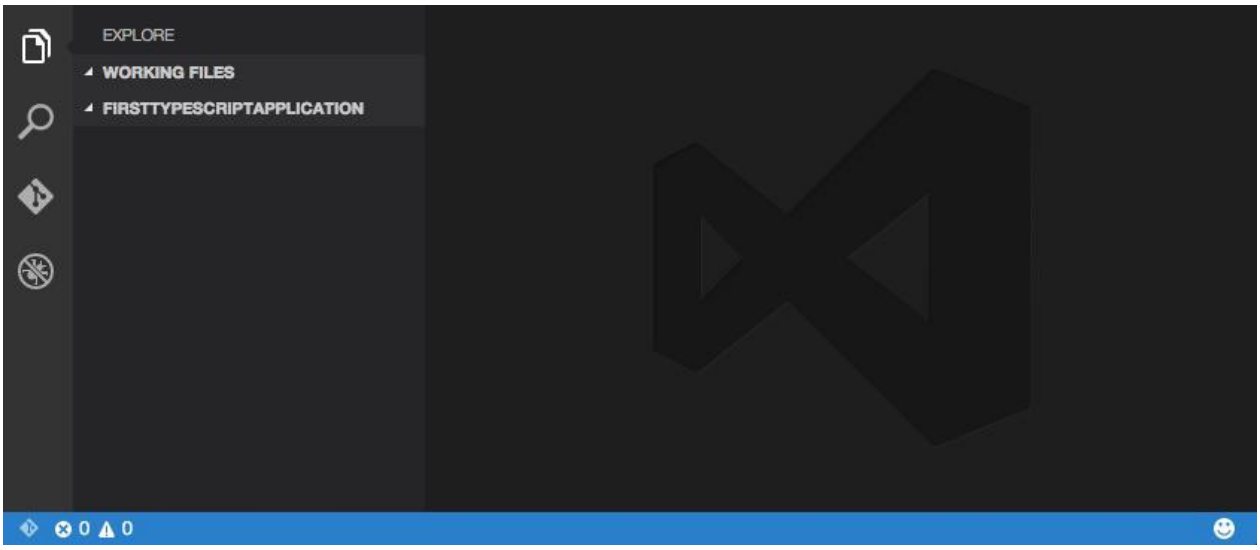
5.2.2 章の手順と同様にしてインストールを行うことができます。

5.3. TypeScript アプリケーションの構築

ターミナル、もしくはコマンドプロンプトを起動し、適当なフォルダで以下のコマンドを実行し、ディレクトリを作成します。

```
mkdir FirstTypeScriptApplication
```

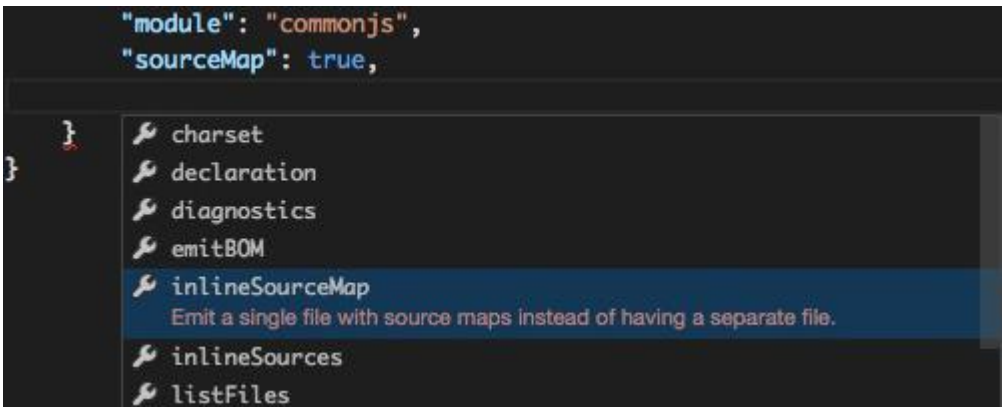
Visual Studio Code を開き、作成した" FirstTypeScriptApplication"ディレクトリを開きます。



まずは TypeScript プロジェクト用の設定ファイルを作成していきましょう。
tsconfig.json を作成し、内容を以下のように変更します。

```
{
    "compilerOptions": {
        "target": "ES3",
        "module": "commonjs",
        "sourceMap": true
    }
}
```

この時、json 設定ファイルに対しても IntelliSense による補完が行われていることが確認できます。



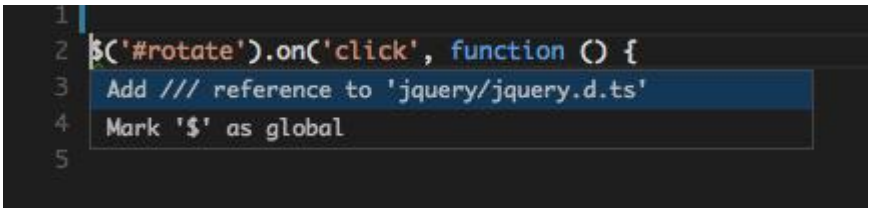
続いて index.html を作成し、ファイルの内容を以下のように変更します。

```
<!DOCTYPE html>
```

```
<html lang="en" ng-app="myApp">
  <head>
    <meta charset="UTF-8">
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js">
</script>
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css" r
el="stylesheet">
    <script src="main.js"></script>
    <title>Hello World</title>
  </head>
  <body>
    <div class="container">
      <div class="row">
        <h1>Hello from AngularJS</h1>
        <div ng-controller="helloWorldController">
          <div class="input-group">
            <input type="text" class="form-control" ng-
model="name" placeholder="Input your name...">
          </div>
          <hr>
          <div class="alert alert-info" role="alert">
            <span class="glyphicon glyphicon-ok" aria-
hidden="true"></span>{{greeting}} {{name}}</div>
          <hr>
          <p>
            <button type="button" class="btn btn-default" ng-
click="hello()">Hello</button>
            <button type="button" class="btn btn-default" ng-click="hi()">Hi</button>
            <button type="button" class="btn btn-default" ng-
click="hey()">Hey</button>
            <button type="button" class="btn btn-success" ng-
click="call()">Call</button>
          </p>
        </div>
      </div>
    </div>
  </body>
</html>
```

続いて TypeScript の開発に移る前に、型定義ファイルの自動インポート機能をご紹介します。

通常 Visual Studio Code が外部ライブラリの呼び出しを検知した際には、右図のような電球マークを表示し（Code Action）、ユーザーに型定義ファイルをインポートする操作を提示します。これにより Visual Studio Code は、DefinitelyTyped リポジトリからそのライブラリの TypeScript の型定義ファイルを自動でインポートし、IntelliSense をライブラリの持つ型やメソッドに対応させることができます。

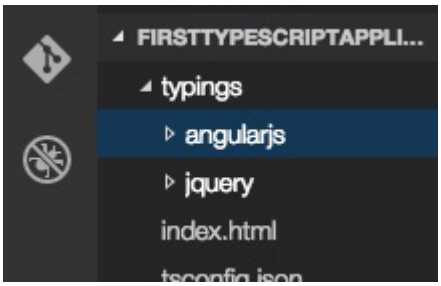


しかしながら現在 Visual Studio Code はプレビュー版であり、TypeScript 開発においては、しばしばその提示が表示されない場合もあります。今回は型定義ファイル用のパッケージマネージャーである tsd（TypeScript Definition manager for DefinitelyTyped）を用いて、型定義ファイルのダウンロード方法をご紹介します。

ターミナルを開き、作成したディレクトリに移動した後、以下のコマンドを実行します。

```
tsd query angular jquery --action install
```

以上で typings ディレクトリが作成され、AngularJS と jQuery の型定義ファイルがダウンロードされました。



その他 tsd の詳細に関しましては、以下のドキュメントを参照してください。

DefinitelyTyped/tsd

それでは Visual Studio Code に戻り、main.ts を作成します。
ファイルの内容を以下の通り変更します。

```
/// <reference path="typings/angularjs/angular.d.ts"/>

module HelloWorld {
  export interface Scope extends ng.IScope {
    name: string;
    greeting: string;
    hello: () => void;
    hi: () => void;
    hey: () => void;
    call: () => void;
  }
  export class Controller {
    constructor($scope: Scope) {
      $scope.name = 'World';
      $scope.greeting = 'Hello';
      $scope.hello = () => $scope.greeting = 'Hello';
      $scope.hi = () => $scope.greeting = 'Hi';
      $scope.hey = () => $scope.greeting = 'Hey';
      $scope.call = () => alert($scope.greeting + " " + $scope.name + "!");
    }
  }

  angular.module("myApp", [])
    .controller("helloWorldController", ["$scope", Controller]);
}
```

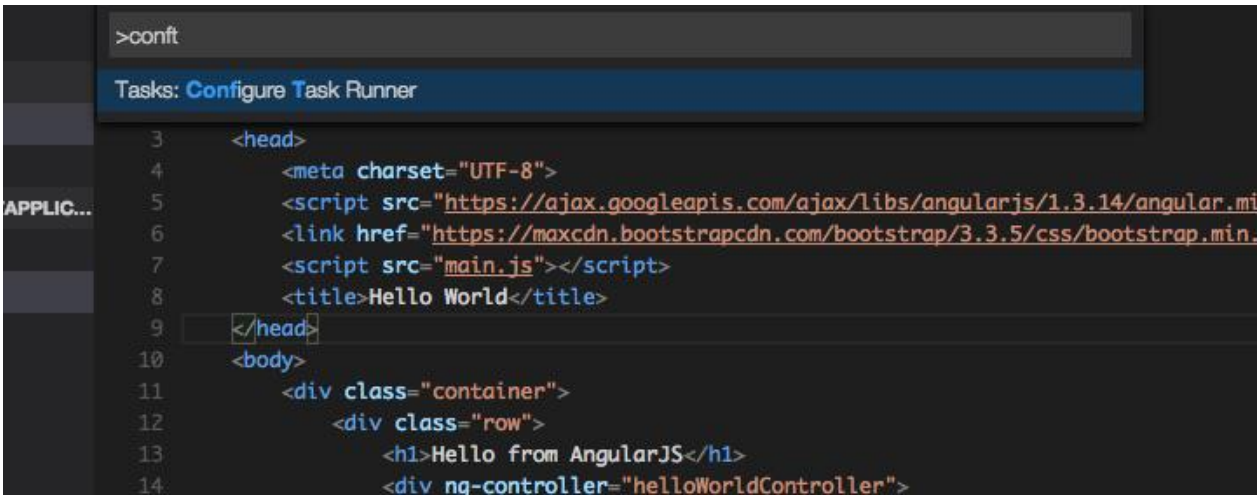
以上でアプリケーションの構築が完了しました。

5.4. タスクランナーを用いた TypeScript アプリケーションのコンパイルと監視

続いて TypeScript で書かれたコードを JavaScript に変換し、実際にブラウザで動作するようコンパイルを行います。Visual Studio Code にはタスクランナーと呼ばれる機能があり、ビルドや最適化などの処理をあらかじめ登録しておくことで、コマンドパレットから簡単に呼び出すことができます。

今回はタスクランナーを用いて、build タスク（TypeScript ファイルの一括コンパイルを行う処理）と watch タスク（ファイルが変更されたかをリアルタイムに確認し、もし変更された場合にはコンパイルを行う）を登録します。

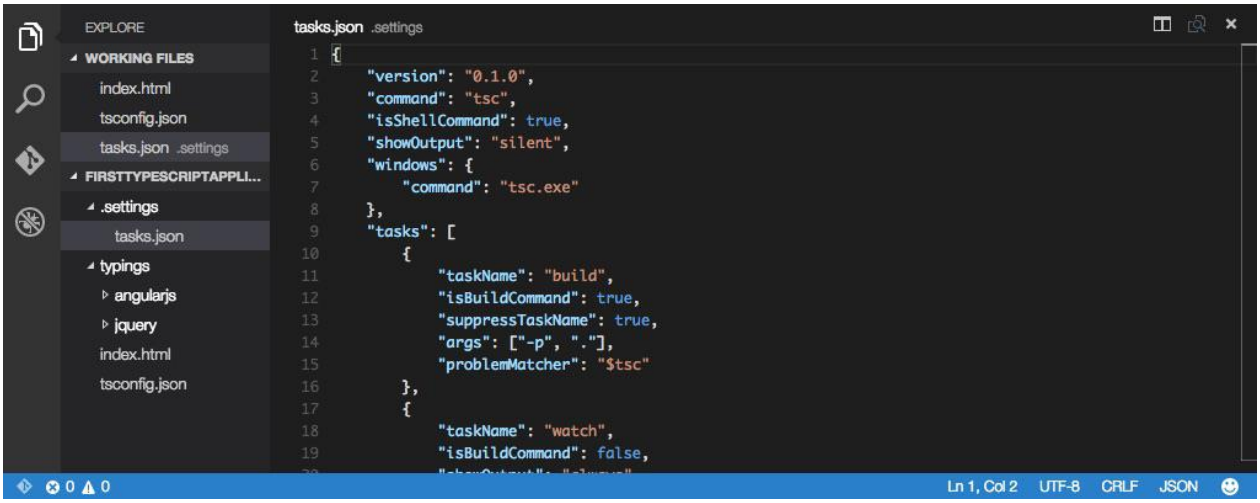
⇧⌘P（Windows は Shift+Ctrl+P）キーを同時に押してコマンドパレットを開き、[Tasks: Configure Task Runner]を選択します。



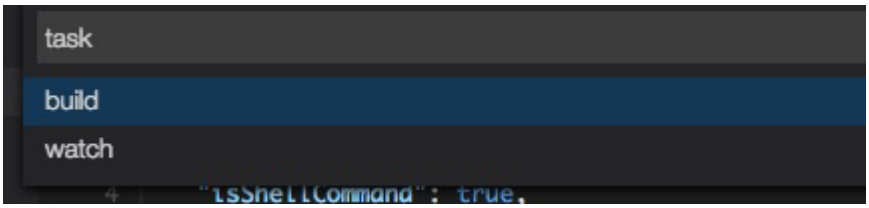
表示されたファイル（tasks.json）の内容をすべて消去し、以下の通り書き換えます。

```
{
    "version": "0.1.0",
    "command": "tsc",
    "isShellCommand": true,
    "showOutput": "silent",
    "windows": {
        "command": "tsc.exe"
    },
```

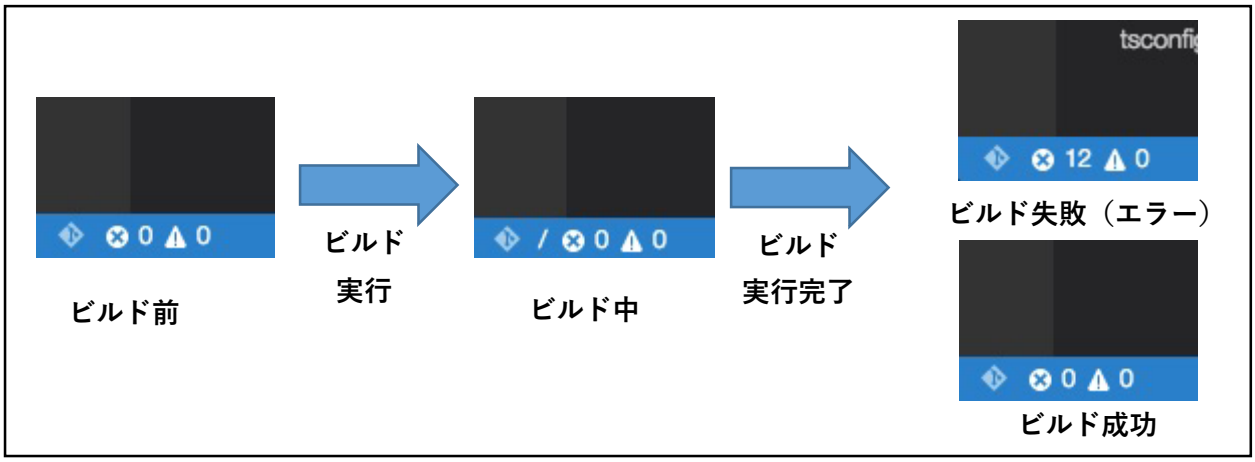
```
    "tasks": [
        {
            "taskName": "build",
            "isBuildCommand": true,
            "suppressTaskName": true,
            "args": ["-p", "."],
            "problemMatcher": "$tsc"
        },
        {
            "taskName": "watch",
            "isBuildCommand": false,
            "showOutput": "always",
            "isWatching": true,
                        "suppressTaskName": true,
            "args": ["-w"],
            "problemMatcher": "$tsc"
        }
    ]
}
```



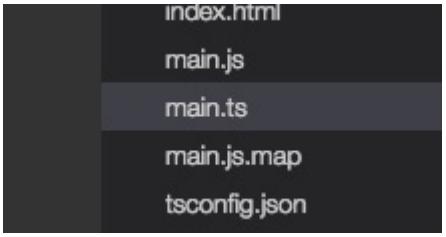
それではまずは build タスクを実行してみましょう。⇧⌘P（Windows は Shift+Ctrl+P）キーを同時に押してコマンドパレットを開き、[Tasks: Run Task]を選択します。その後、登録されているタスクが一覧表示されますので、build を選択します。



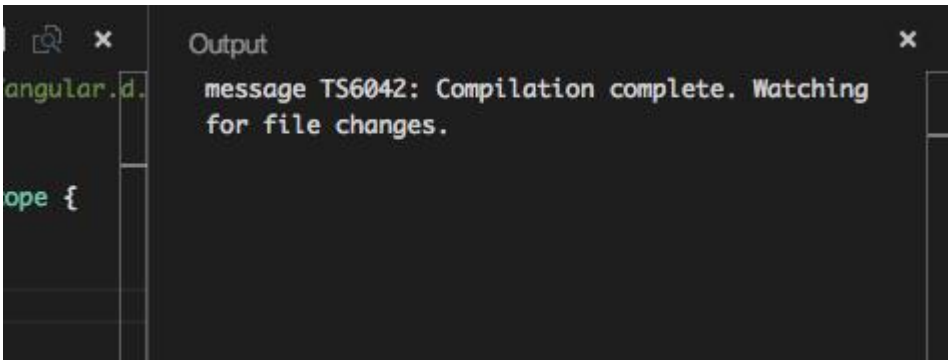
ビルドが実行されます。この時左下のステータスバーに注目してください。ビルド実行中および完了時のステータスは逐次表示されます。



ビルドに成功するとエクスプローラビューに.js ファイルなどが増えていることが確認できます。

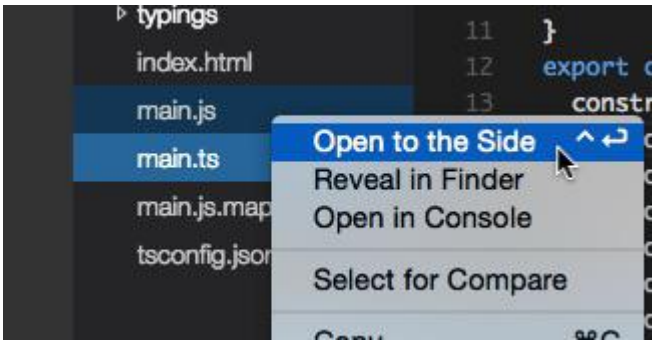


最後に watch タスクを実行してみましょう。
⇧⌘P（Windows は Shift+Ctrl+P）キーを同時に押してコマンドパレットを開き、[Tasks: Run Task] を選択します。その後、登録されているタスクが一覧表示されますので、watch を選択します。

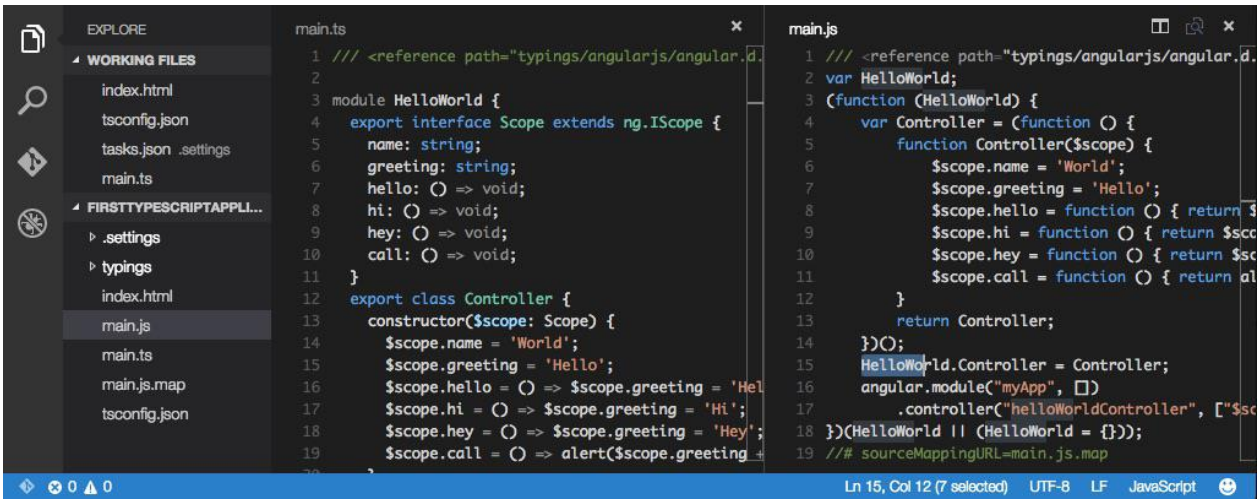


画面が自動的に分割され、Output パネルに"Watching for file changes."と表示されたら、タスクが起動しています。

では TypeScript ファイルを開きましょう。(ここでは main.ts とします) さらに同様にしてエクスプローラビューから main.js (main.ts の生成物) を右クリックし、[Open to the Side]を選択します。



右画面に main.js が表示されます。



この状態で左画面のソースコードを変更し保存すると、右画面でそのコンパイル結果がリアルタイムに反映されていることが確認できます。

watch タスクはこの間、バックグラウンドで動作し続けています。終了するには、⇧⌘P（Windows は Shift+Ctrl+P）キーを同時に押してコマンドパレットを開き、[Tasks: Terminate Running Task] を選択します。

5.5. TypeScript アプリケーションのデバッグ

現状、Visual Studio Code のデバッグは Node.js または mono ベースのみのアプリケーションに対応しています。Node.js のデバッグの詳細は 3.4 章を参照してください。
クライアントサイドの TypeScript・JavaScript デバッグは、ご利用の各ブラウザのデバッグツールを用いる必要があります。

6. Visual Studio Code と Git で行うバージョン管理

6.1. バージョン管理システムを用いるメリット

バージョン管理システムはファイルの変更状態を管理し、過去の状態の復元や比較を容易に行うことができるシステムです。また単純にバックアップを取るのではなく、差分情報のみを記録しているため、比較的少ない容量で管理を行うことができます。

また複数人で同時に作業を行う場合を考えてみましょう。同じファイルを同時に編集していた場合（**コンフリクト**）、従来はその統合作業（**マージ**）を手動で行う必要がありました。またファイルが同時に編集されていたことに気付かず、誤って相手の変更を上書きしてしまう可能性もあります。しかしながらバージョン管理システムは常にこの同時編集を検知し、自動的に統合（**自動マージ**）を行うか、修正案を提示します。

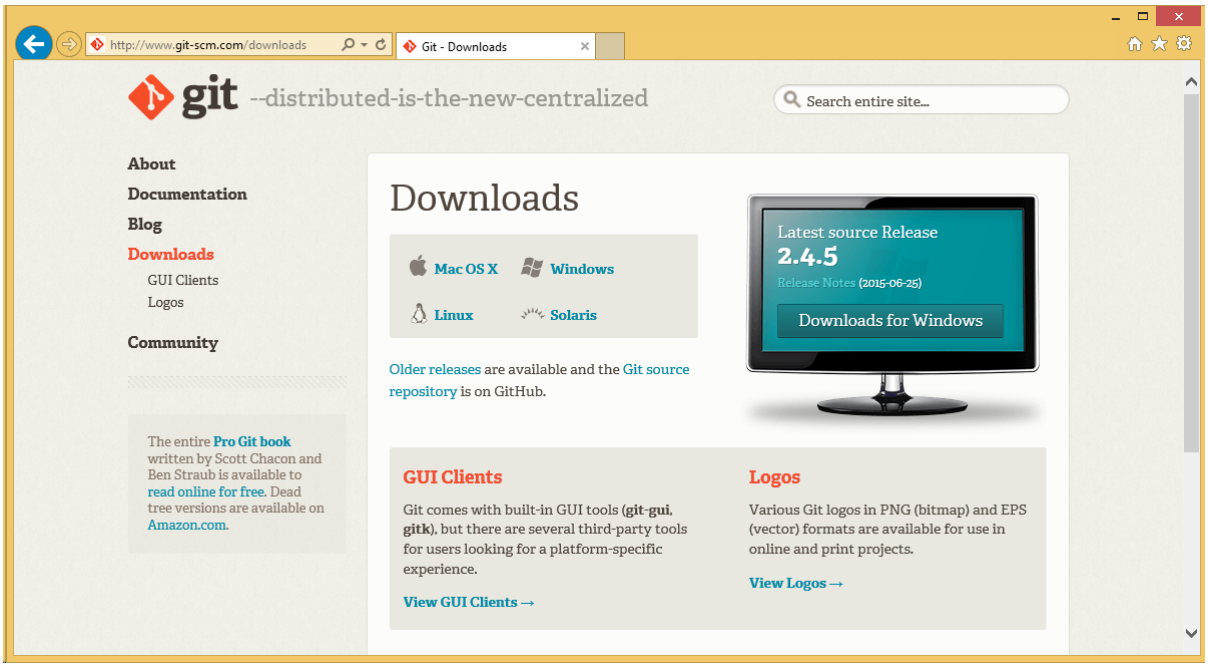
Git は、このバージョン管理システムの一種です。ファイルの変更情報を管理する機構（**リポジトリ**）がローカルとサーバーにあり、それぞれの変更内容を同期させることによって、全体のファイル管理を実現しています。これにより、開発者はネットワークにつながっていなくても、好きな時に変更を記録（**コミット**）でき、また他人の編集した最新ファイルを上書きしようとした場合には、自動的に統合されるか警告が出される、という特徴があります。

この章では Git を用いたプロジェクト・ファイルの管理方法をご紹介します。

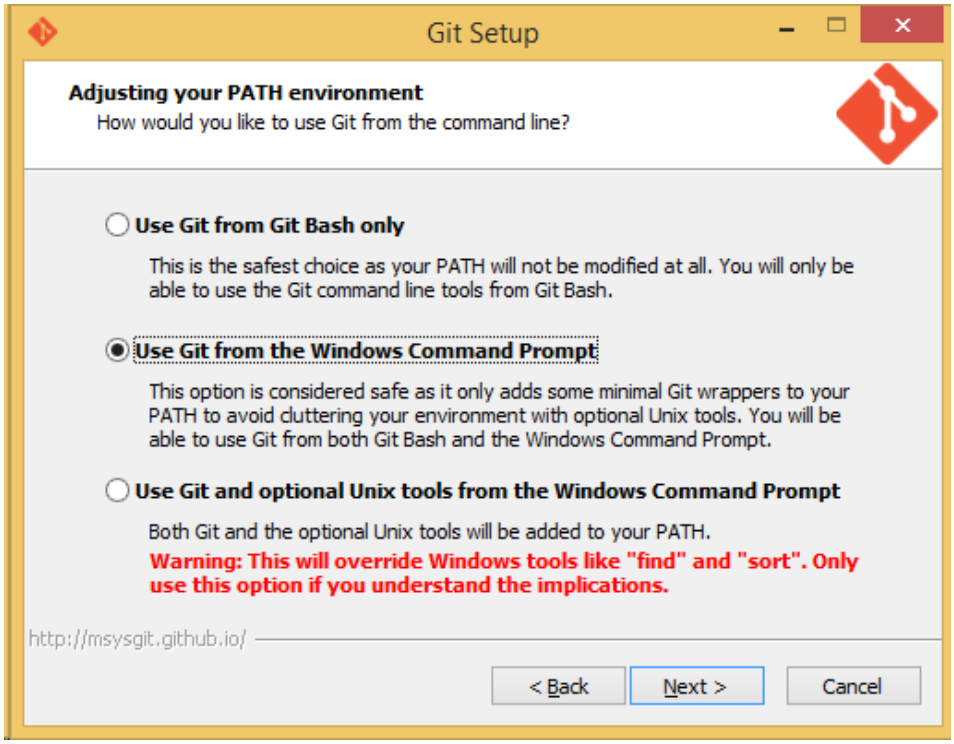
6.2. Git のインストール

6.2.1. インストール方法（Windows 編）

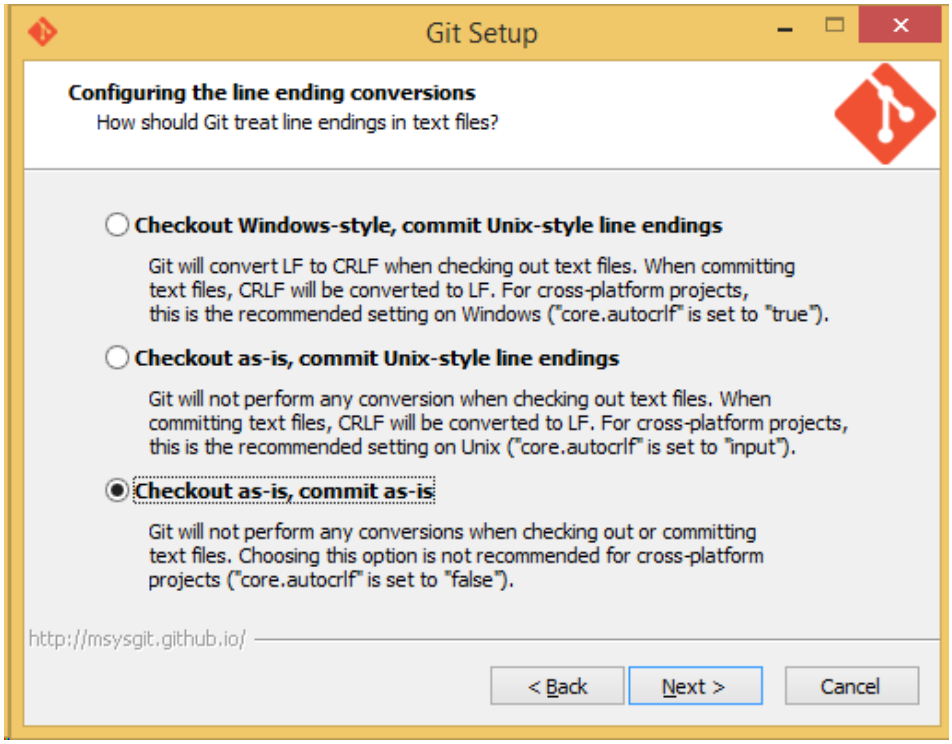
1. http://www.git-scm.com/downloads にアクセスします
2. Windows をクリックします。(2015 年 7 月現在、Git-1.9.5-preview20150319.exe がダウンロードされます)



3. インストーラの手順に従って使用許諾契約の同意とインストールを進めます。インストール中に以下の画面が表示された際には、"Run Git from the Windows Command Prompt"を選択することをお勧めします。



また Visual Studio Code は改行コードとして、標準で LF および CRLF に対応しています。そのため Git コマンドによりチェックアウト/コミット時の改行コード変換機能を利用しなくても正常に編集することができます。以下のような改行コード選択画面が表示された際には、”Checkout as-is, commit as-is”を選択することをお勧めします。



4. インストールを先に進めていき、インストールを完了させます。

続いて Git を利用するための初期設定を行います。

5. コマンドプロンプトを開き、以下のコマンドを実行します。（カッコ内はご自身の情報に置き換えてください）

```
git config --global user.name “(Your Name)”
git config --global user.email “(Your email)”
```

6. 以上で Git の設定が完了しました。

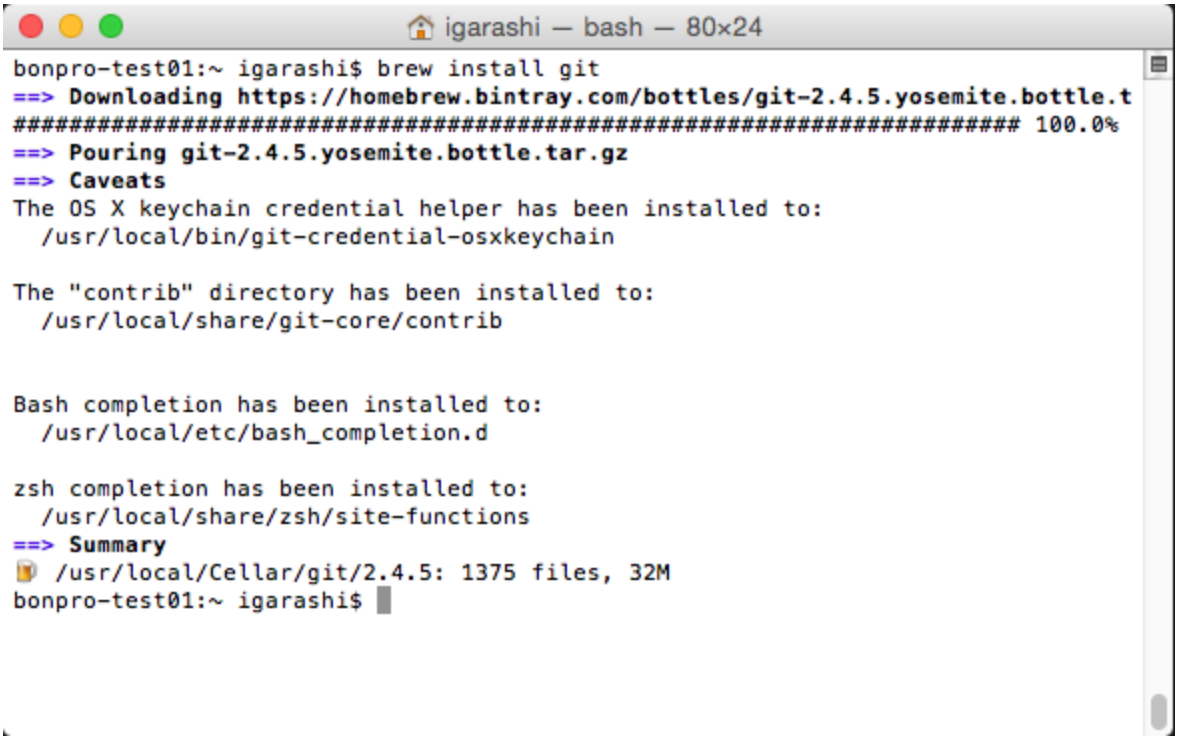
6.2.2. インストール方法（Mac 編）

6.2.1 章の方法以外に、4.3.2 章で導入した Homebrew を用いてインストールすることができます。

ここでは Homebrew を用いた Git のインストール方法をご紹介します。

7. ターミナルを開き、以下のコマンドを実行します。

```
brew install git
```



8. 以上で Git のインストールが完了しました。

続いて Git を利用するための初期設定を行います。

9. 以下のコマンドを実行します。（カッコ内はご自身の情報に置き換えてください）

```
git config --global user.name “(Your Name)”
git config --global user.email “(Your email)”
```

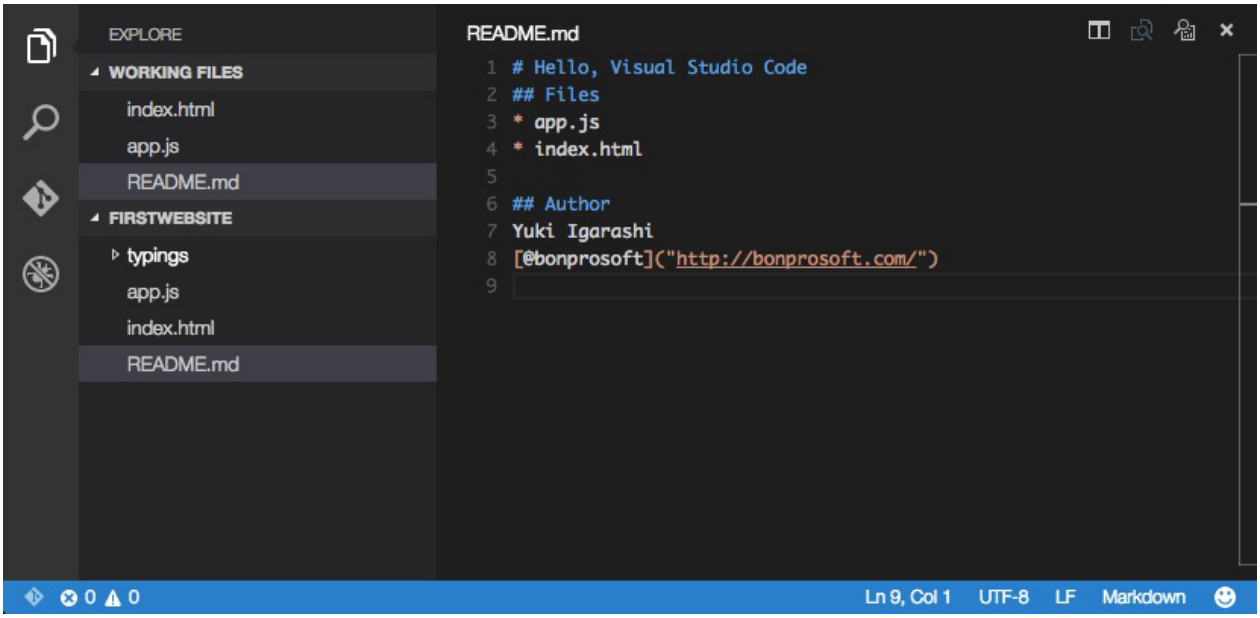
10. 以上で Git の設定が完了しました。

6.2.3. インストール方法（Linux 編）

Linux の場合、6.2.1 章の方法以外にも、お使いのパッケージマネージャー（yum や apt-get など）を利用してインストールすることも可能です。詳細はお使いのディストリビューション向けの git インストールマニュアルをご覧ください。

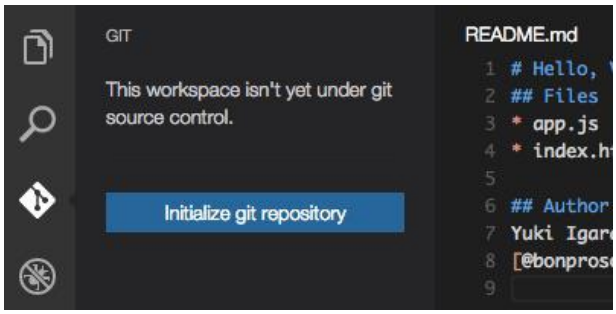
6.3. Visual Studio Code と Git を用いたソースコード管理の実践

ソース管理を行いたい適当なプロジェクトを用意し、Visual Studio Code で開きます。ここでは 2 章で作成した簡単な Web サイトを例に扱います。

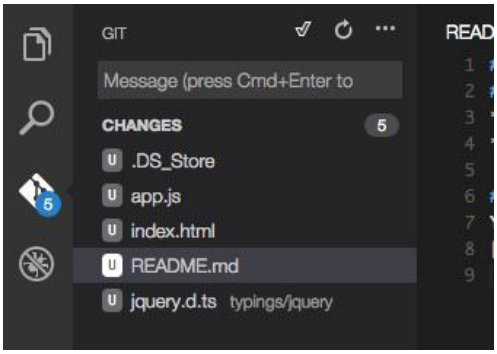


画面左側のビューバーから Git ボタンを選択します。

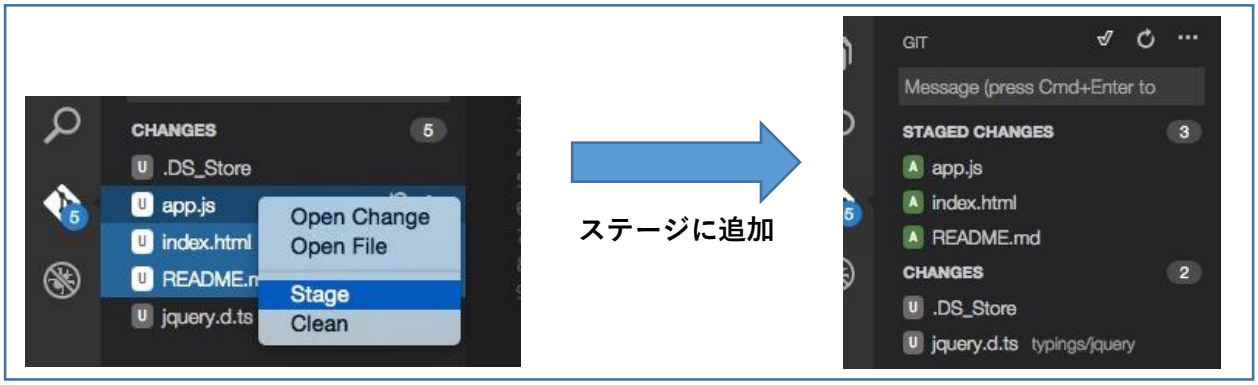
開いているディレクトリに Git リポジトリがない場合（Git の管理下にない場合）は、右図のような画面が表示されます。[Initialize git repository]をクリックし、このディレクトリに対して Git リポジトリを作成します。



Git リポジトリの作成が完了すると、Git ビューには現在のファイルの変更状況が表示されます。右図には新規に追加されたファイルとして 5 つのファイルがあることを示しています。

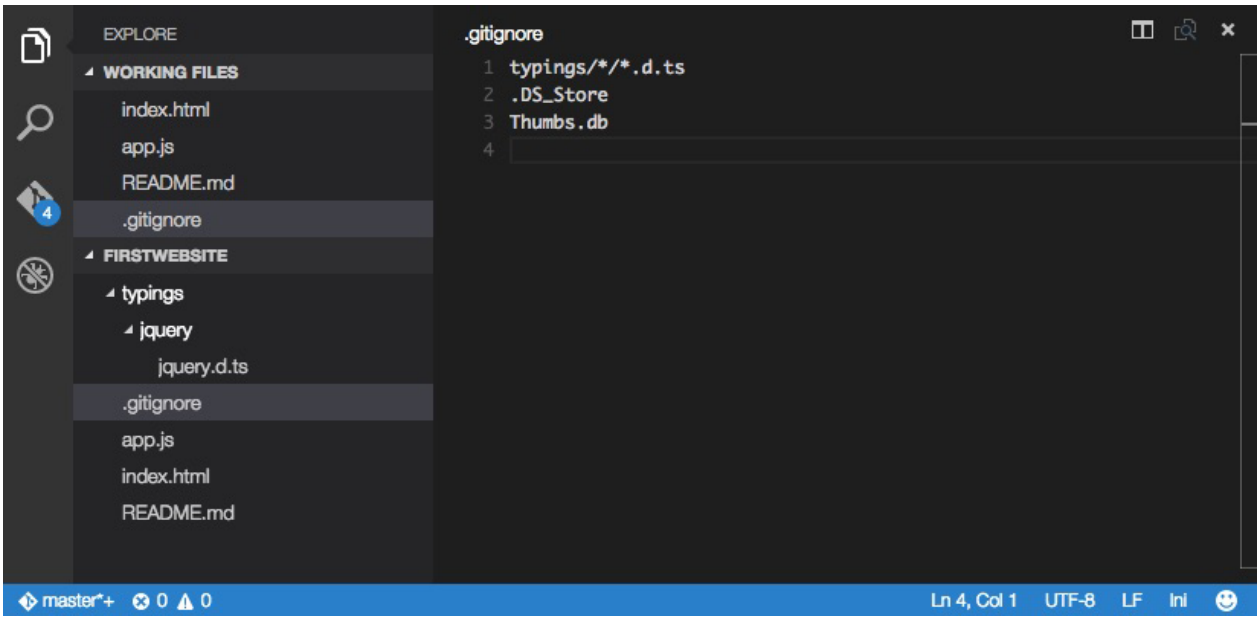


次にファイルをステージに追加します。(ステージに追加することで、ファイルを Git リポジトリ管理下に置き、追加することができます) ⇧キーまたは⌘キー（Windows では Shift キーまたは Ctrl キー）を押しながら、*app.js* と *index.html* と *README.md* の 3 つのファイルを選択し、右クリックします。コンテキストメニューから[Stage]をクリックすると、これらの 3 つのファイルをステージに追加します。



残りの 2 つのファイルである.DS_Store と jquery.d.ts については OS によって作成された不要なファイル、もしくは開発時にのみ必要な外部ファイルであり、リポジトリには追加する必要はありません。これ以外にも、コンパイル済みのバイナリやパッケージマネージャーによって追加されたライブラリをリポジトリに追加すると、使用しているライブラリがアップデートされたときに、そのアップデート内容を反映するためのコミットが必要になるほか、環境固有の情報（ビルド時のパス）などを含むデータが外部へ公開される可能性があるため、不要なファイルは極力リポジトリに追加しないことをおすすめします。Git では*.gitignore* ファイルを用意することによって、これらの不要なファイルをリポジトリから自動的に除外することができます。

ビューバーからエクスプローラを選択し、新規作成ボタンを押して.gitignore ファイルをディレクトリのトップに作成します。

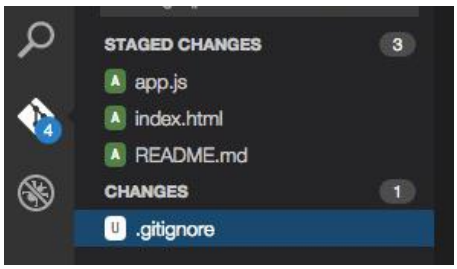


内容を以下のように変更します。

```
1 typings/*/*.d.ts
2 .DS_Store
3 Thumbs.db
```

1 行目は typings ディレクトリにある拡張子 d.ts を持つファイルを無視することを意味します。
2 行目は.DS_Store（Mad OS X が作成するメタデータ）を無視することを意味します。
3 行目は Thumbs.db（Windows が作成するメタデータ）を無視することを意味します。

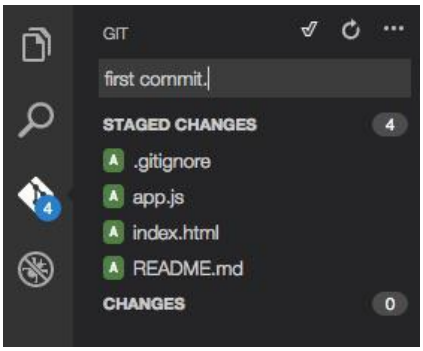
ファイルを保存して Git ビューに戻ると、先ほどまであった不要なファイルが一覧から消えていることが確認できます。



.gitignore はこのリポジトリに対する設定ですので、他の端末で作業したときにも同様の設定が適用されるよう、同様の手順でリポジトリに追加しましょう。

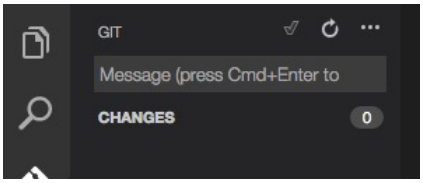
それでは変更点をコミット（保存）しましょう。

Git ビューの上部にあるテキストボックスにコミットメッセージ（今回のコミットに対する説明）を入力します。このテキストボックスは複数行の入力にも対応していますので、3 行目に詳細なコミットメッセージを記入することも可能です。

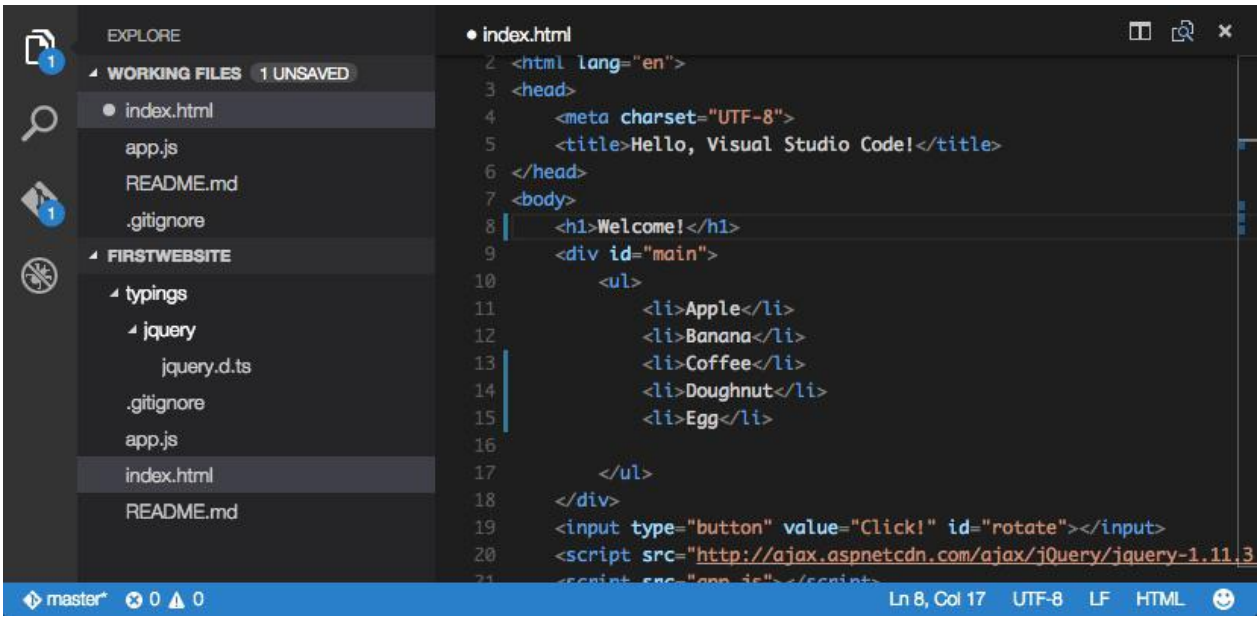


入力が完了したらチェックボックスをクリックするか、⌘+Enter キー（Windows では Ctrl+Enter）を押すと、コミットが行われます。

コミットが正常に終了すると、ステージされていた変更が保存され、STAGED CHANGES のアイテムが空になります。

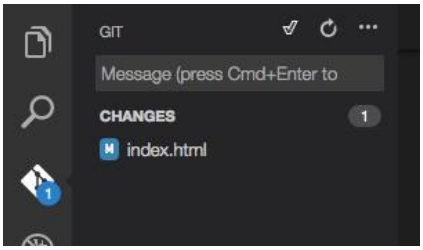


ではエクスプローラビューに戻り、例として index.html を編集してみましょう。
この時、行番号左側に注目してください。青いインジケータが表示され、コーディング中にどの部分が変更されたかが一目で把握できるようになっています。

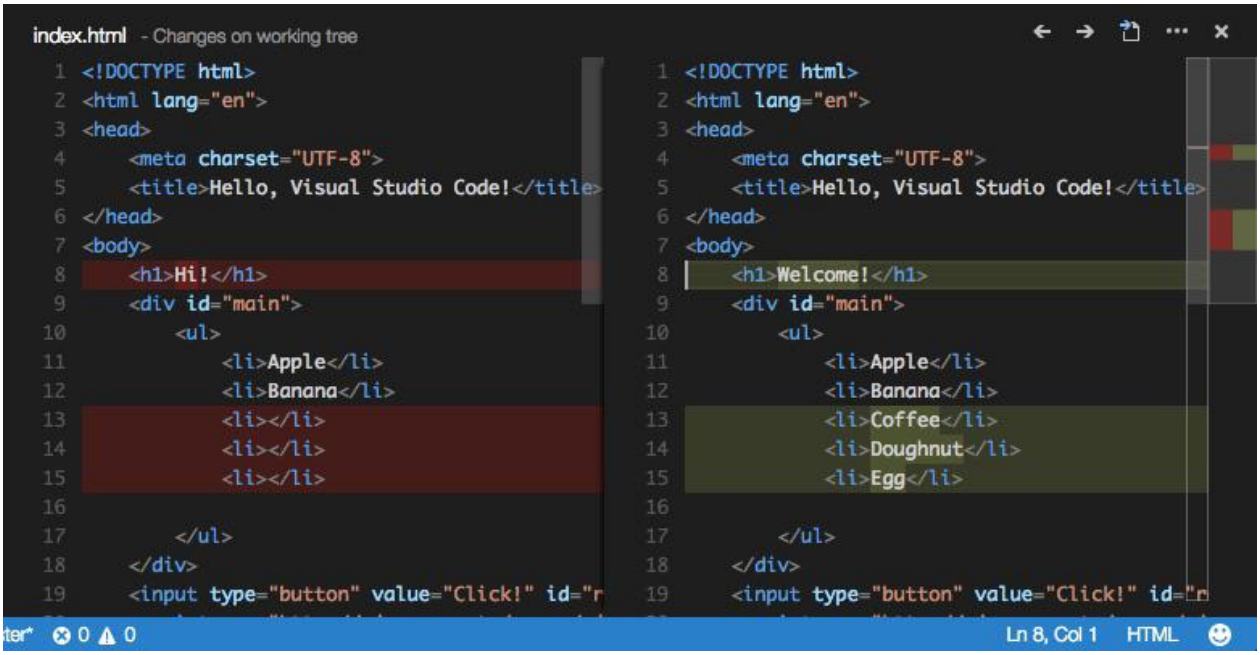


このようにファイルが Git リポジトリ管理下にあると、Visual Studio Code がそれを検知し、エディタと連携していることが分かります。

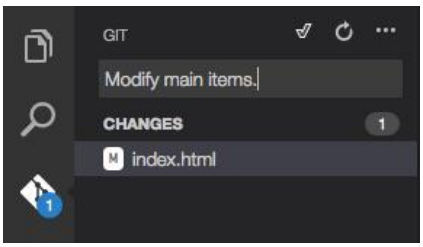
では再度 Git ビューに戻りましょう。すると Git ビューの中に index.html が変更点として追加されていることが分かります。このアイテムを右クリックし、コンテキストメニューから[Open Change]をクリックします。



すると右側のエディタパネルが自動的に左右に分割され、前回の Git コミット時と現在のファイルの変更点をマーカーで表示されます。



再度コミットメッセージを入力し、コミットを行います。コミットが正常に行われると、再び変更点がクリアされます。



6.4. リモートリポジトリとの連携と GitHub 用いた Azure への自動デプロイ

Git はローカルリポジトリと外部のサーバーにあるリポジトリとを同期させることで、さらに高度な開発を行うことができます。例えば複数人での同時作業・ファイル管理はもちろん、Azure などのサービスと連携させることで、サーバーとの同期時に自動的にデプロイなども行うこともできます。

Microsoft は Visual Studio Online を提供しており、このサービスを用いることでソース管理・要件やバグのトラッキング、クラウドでのアプリケーションのビルド、アプリケーションの動的レポートな

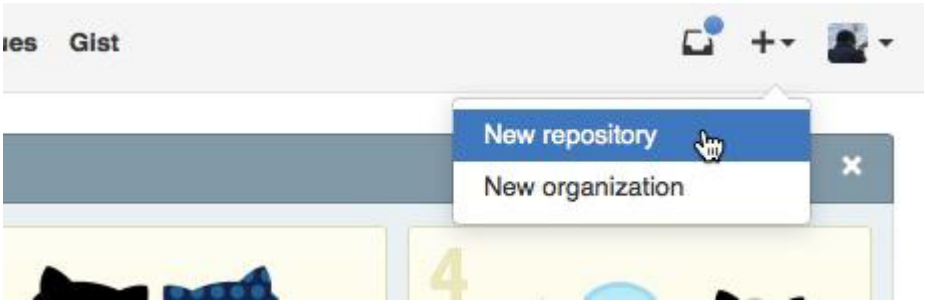
ど、プロジェクト開発に必要な様々なサービスを利用することができます。Visual Studio Online は 5 ユーザーまで無料で使用できます。詳細は以下のリンクを参照してください。

https://www.visualstudio.com/ja-jp/products/what-is-visual-studio-online-vs.aspx

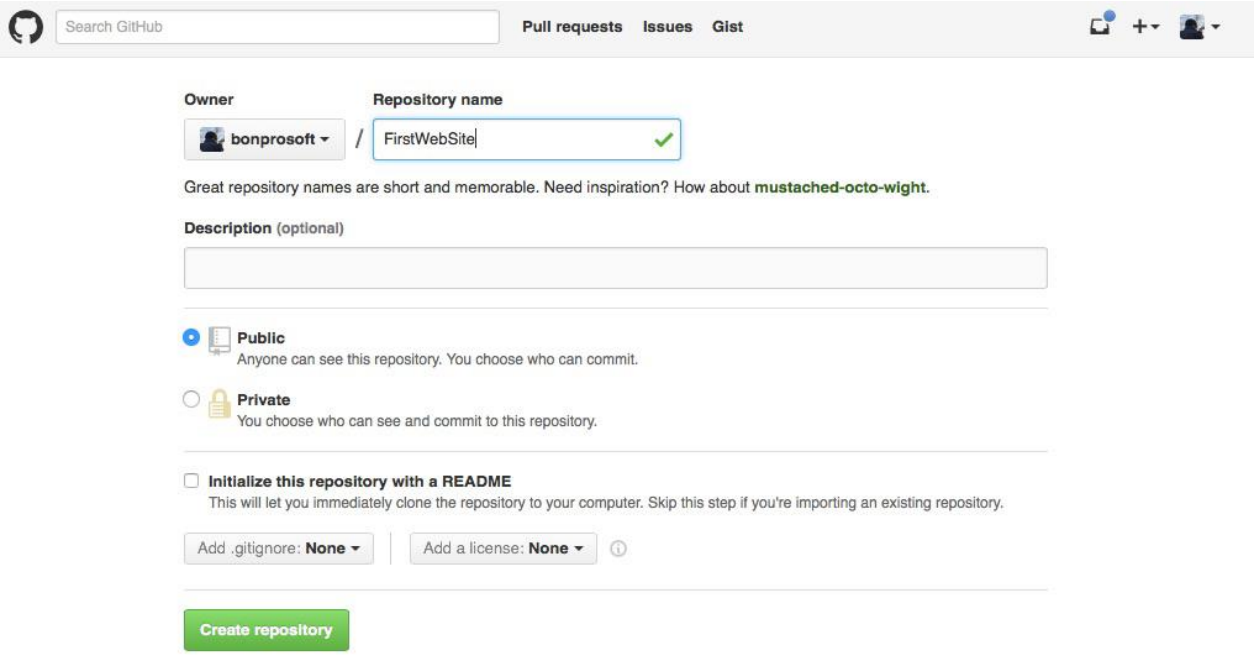
この章では外部サーバーとして Visual Studio Online ではなく、一般的な開発に広く用いられている GitHub を取り上げます。GitHub でホストしているリポジトリとの同期方法、および GitHub と Microsoft Azure の連携機能を利用した、自動デプロイの方法についてご紹介します。

なお、この章では GitHub の登録方法、GitHub サーバーとの認証キーの登録方法、および Azure の登録方法は取り上げませんので、ご了承ください。

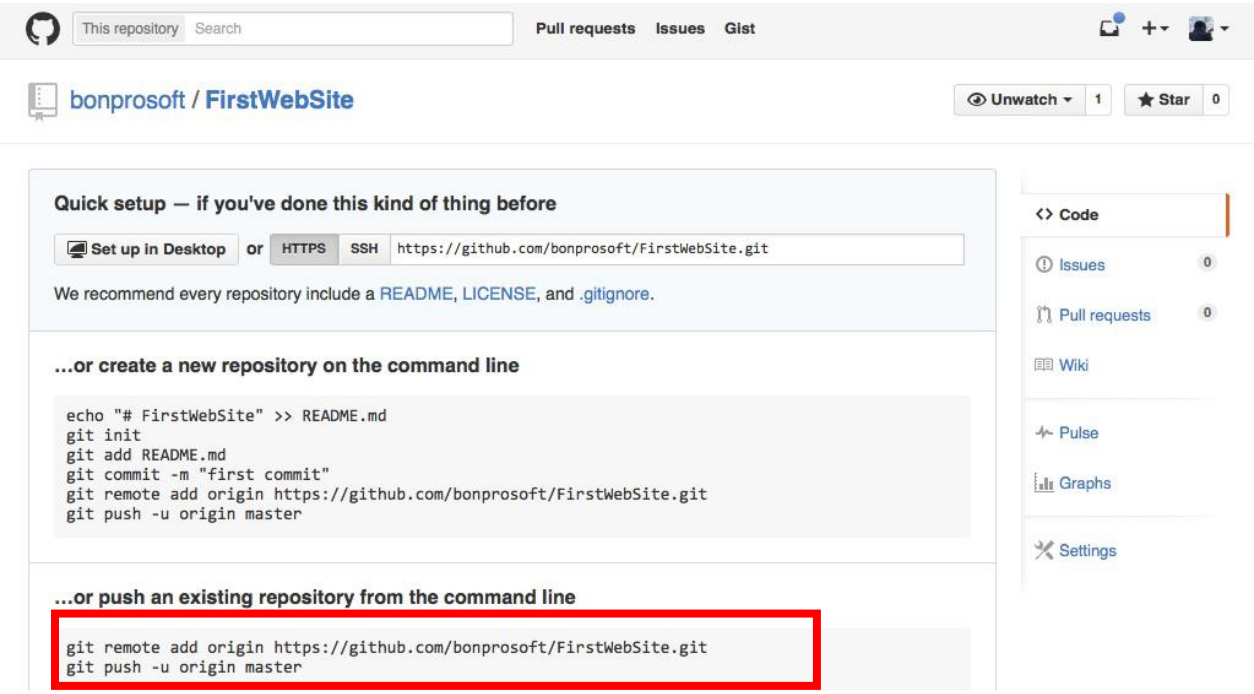
はじめに GitHub にログインし、右上のプラスボタンからリポジトリを新規作成します。



リポジトリの新規作成画面では Repository name に適当な値を入れ、Create repository ボタンをクリックします。

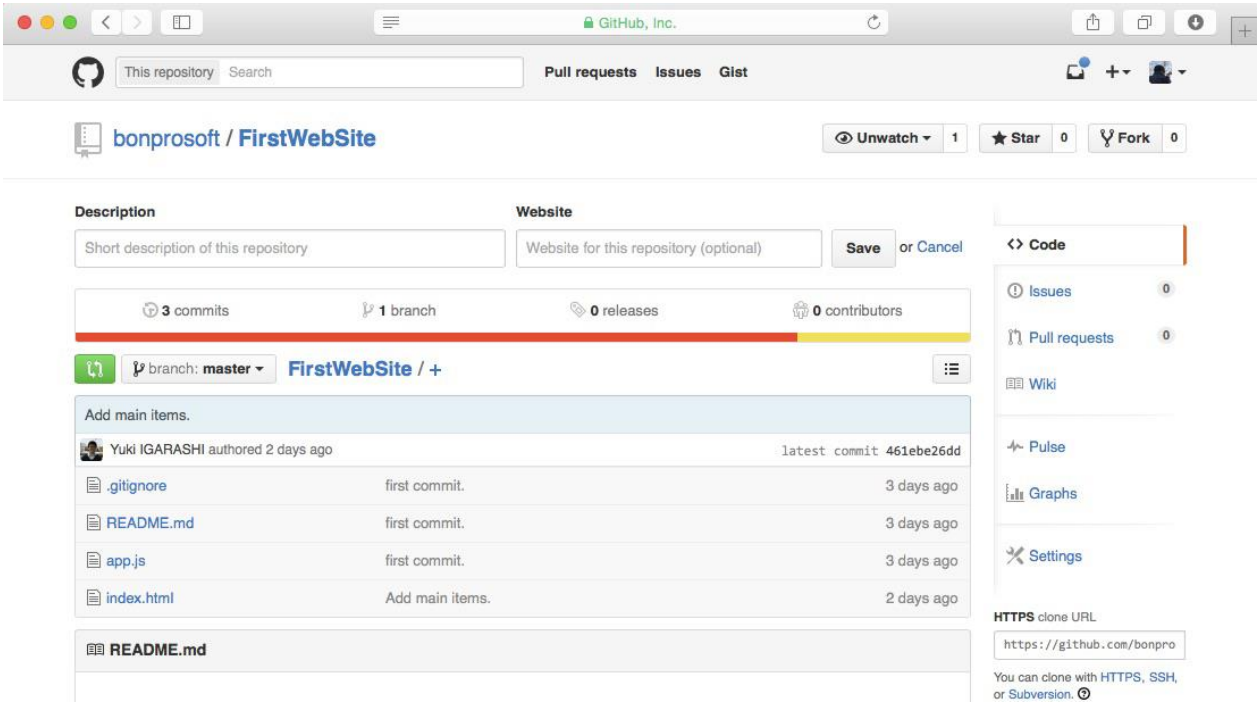


リポジトリが作成されると、下記のような Quick setup 画面が表示されます。



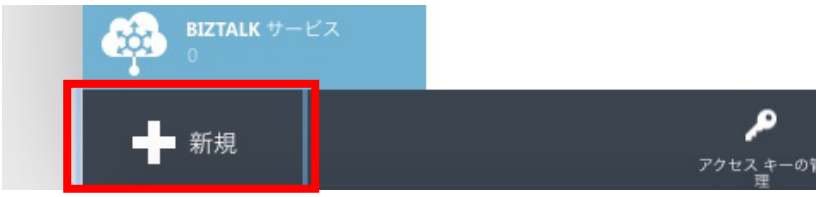
ターミナルに戻り、ローカルリポジトリのディレクトリの中で、Quick setup 画面で表示された"...or push an existing repository from the command line"にあったコマンドを実行します。

途中で認証情報を求められた場合は、GitHub の認証情報を入力してください。
以上でリモートリポジトリが登録され、初めての同期が行われました。
ブラウザをリロードすると、ローカルリポジトリのコミット内容がサーバーに反映されていることが確認できます。



次に Azure の Git の連携を行いましょう。下記 URL から Azure クラシックポータルにアクセスします。

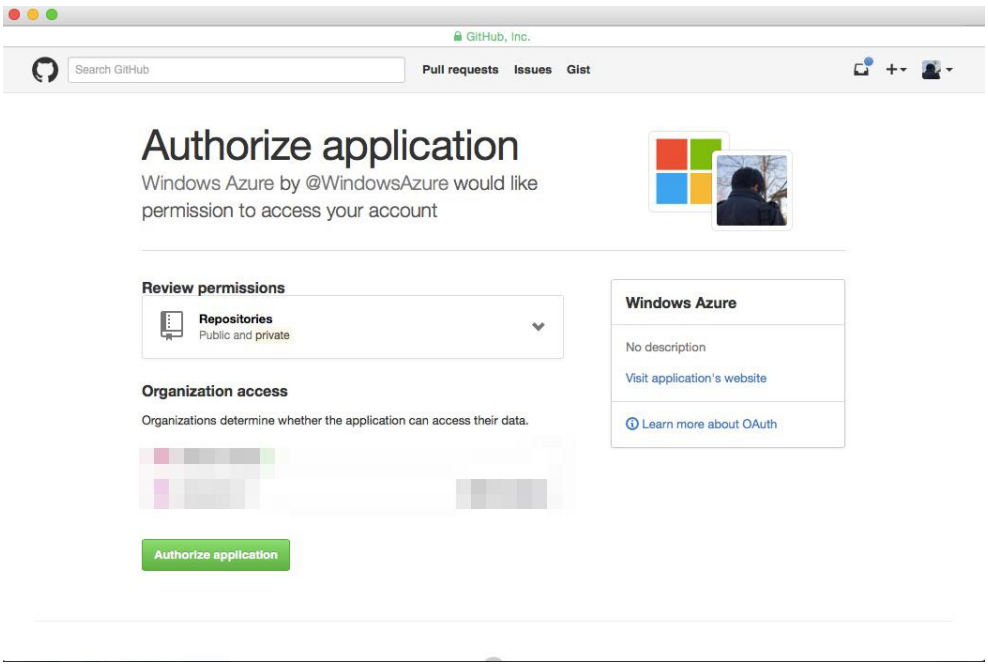
https://manage.windowsazure.com/



左下にある[新規]ボタンから、[COMPUTE]、[WEB アプリ]、[カスタム作成]の順にクリックします。

URL には適当な名前を入力し、[ソース管理から発行]チェックボックスを選択します。
右下の矢印ボタン（次へボタン）をクリックします。

この画面ではソース管理の発行元の種類を選択します。今回は GitHub からのデプロイですので、GitHub を選択し、次へ をクリックします。

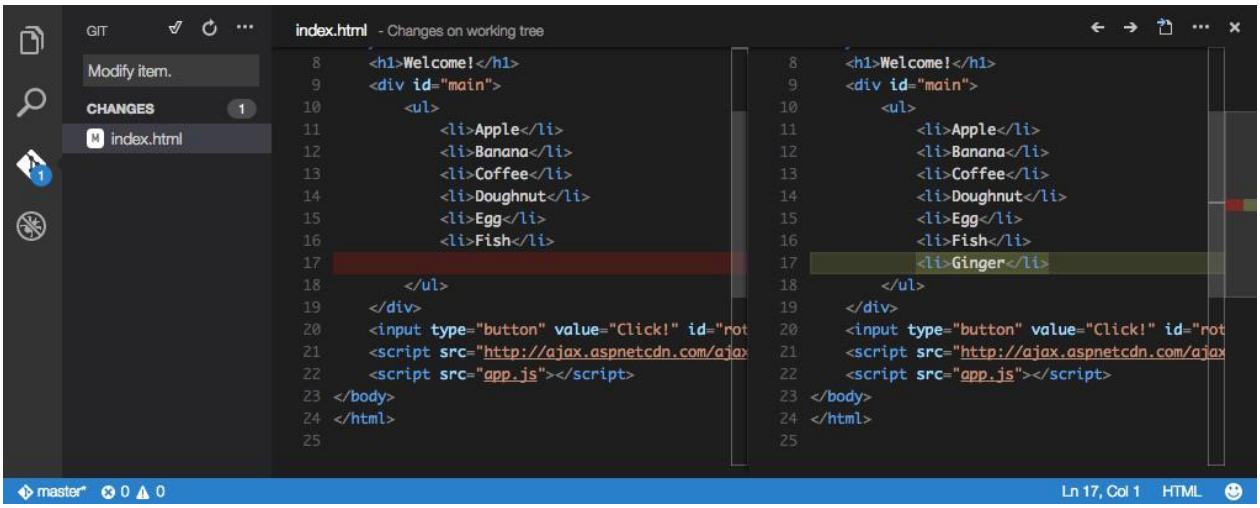


Azure が GitHub のリポジトリ情報を扱うには GitHub との連携が必要となります。上記のような許可を求める画面が表示されたら[Authorize Application]ボタンをクリックします。

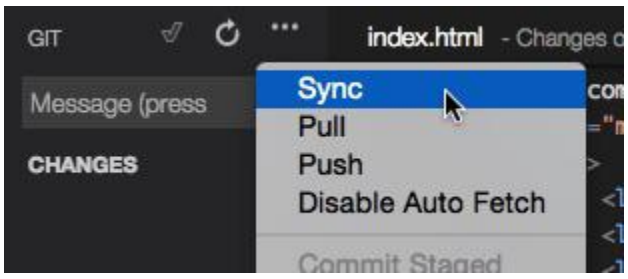
続いてデプロイするリポジトリの選択画面では、先ほど作成したリポジトリの名前を選びます。デプロイする分岐にはフックするブランチ名を入力します。今回の場合は master ブランチのみ扱いますので、リポジトリ名のみを選択して完了ボタンをクリックします。

ウェブサイトの作成が完了すると、一覧での状態が[実行中]に変わります。
それでは URL 列にあるリンクをクリックして、作成されたウェブサイトにアクセスしてみましょう。

正常に GitHub リポジトリからデータが取得され、Azure サーバーへ展開されていることが確認できました。
では Visual Studio Code でローカルリポジトリを開き、index.html に変更を加えてみましょう。



例えば、上の画像のようにリストに要素を 1 つ加えます。
この変更を Commit して保存します。



ではサーバーとの同期を行い、今回の Commit をサーバーへ反映させましょう。
Git ビューの右側にある[...]ボタンをクリックし、[Sync]を押します。Sync 中は Git ビューに青いインジケータが表示され、Sync が完了するとインジケータが消えます。

では再度ウェブサイトを確認してみましょう。

先ほどの変更がウェブサイトにも適用されていることが確認できます。
この背景として、GitHub サーバーとの Sync 時に master ブランチの更新があったため、GitHub サーバーから Azure サーバーへフックが行われ、Azure サーバーが自動的に最新の情報を取得・展開が行われています。

7. 参考リンク集

- Visual Studio Code
  https://code.visualstudio.com/

- Documentation for Visual Studio Code（英語）
  https://code.visualstudio.com/Docs

- Visual Studio 製品ページ
  https://www.visualstudio.com/

- MSDN Blog : Visual Studio Code
  http://blogs.msdn.com/b/vscode/

- de:code 2015 : 新しいクロス プラットフォーム開発 Visual Studio Code
  https://channel9.msdn.com/Events/de-code/decode-2015/DEV-023

- Build 2015 : Visual Studio Code: A Deep Dive on the Redefined Code Editor for OS X, Linux and Windows （英語）
  https://channel9.msdn.com/Events/Build/2015/3-680

- Twitter : Visual Studio Code
  https://twitter.com/code

8. 付録

8.1. DreamSpark のご紹介

高等学校・専門学校・専修学校・高等専門学校・大学に所属する学生/教職員には DreamSpark と呼ばれる支援プログラムがあります。このプログラムでは、Visual Studio Professional を含む開発ツールやサーバーOS、さらに Microsoft Azure の一部を無料で学生向けに提供しております。（教育機関は 10,800 円からとなります）本ドキュメントでご紹介した Azure Web Apps および Visual Studio Online は DreamSpark でも無料でご利用になれます。さらに Visual Studio などの製品は、一度インストールすれば卒業後も永続的に利用可能となっておりますので、学生の方は是非こちらもご検討ください。

・DreamSpark ウェブサイト
https://www.microsoft.com/ja-jp/education/dreamspark.aspx

・DreamSpark ソフトウェア カタログ
https://www.dreamspark.com/Student/Software-Catalog.aspx