2000年に.NETとC#に関する情報とプレビュー版が公開され、2002年には、Visual Studio .NET 2012製品群の1ラインナップにVisual C#が加わる形で正式リリースされました（この当時は、プログラミング言語ごとに製品が分かれていました。現在はそういう分け方はしていません）。

●C#と.NET

C#を説明するにあたって付随するのが.NETの存在です。C#の話に入る前に、.NETとの関係性について説明しておきます。

一般に、プログラミング言語で書いたプログラムを動かすためには、次のようなものが必要になります。

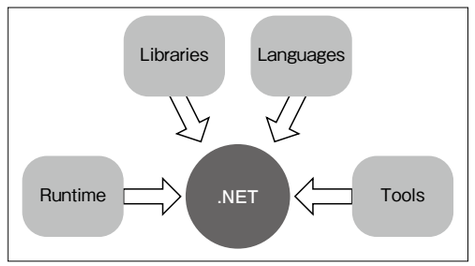
- 実行時環境（Runtime）
- ライブラリ（Libraries）
- 言語コンパイラー（Languages）

また、プログラミングを補助するためのツール（Tools）も、ほぼ必須と言えます。**図2**に示すように、「.NET」というものはこれらRuntime + Labraries + Languages + Toolsの4つを含むものです。

言語コンパイラー

C#はこのうちの一部、Languagesに含まれる言語コンパイラーの1つです。

○**図2　.NETの成り立ち**



C#と.NETというレイヤーが分かれているのは、.NETという枠組みの中で、C#以外にもさまざまなプログラミング言語を実装して、相互にやり取りできるようにするためです。たとえば、VB（バージョン7以降）やF#などのプログラミング言語は.NET上で動き、C#と簡単に相互運用できます。.NETの目的の1つに、複数のプログラミング言語の間のギャップをなくすことも含まれています。

実行環境

プログラムを実行するのに最低限必須なものがあります。これを実行環境（Runtime）と呼びます。

ほとんどのプログラミング言語で、まずメモリ管理などの機能が必須です。また、C#の場合は、一度IL（Intermediate Language）と呼ばれる形式にしたものを、実行時にネイティブコード化してプログラムを動かすので、その解釈プログラムも必要になります。このような、C#を動かすのに必要な実行環境は.NET Runtimeなどと呼ばれます。

ライブラリ

最近のプログラミング環境では、多くの機能を言語機能としてではなく、ライブラリとして提供しています。タッチやマウスなどのユーザーからの入力を受け付け、文字や絵として結果を出力するのも、ライブラリを通してです。

「一度書いたらどこででも動く」を実現するためには、実行環境の規格に加えて、標準で使えるライブラリが何かも定まっている必要があります。.NETの場合は、この実行環境+標準ライブラリの規格を指す言葉が.NET Framework（当初からあるWindows向けのもの）や.NET Core（2015年世代で新たに作られたクロスプラットフォーム向けのもの）になります。

他のプログラミング言語では、これらをあまり区別しなかったりもします。たとえばJavaの場合、実行環境も標準ライブラリも言語コンパイラーもすべてで「規格どおり」に実装し、認定を受けて初めてJavaを名乗れることになります。

●C# 1.0：最初のC#

.NETのような大掛かりなものを一から作っていたわけですから、それだけで相当な労力がかかったはずです。最初のC#、つまり、C# 1.0は、リリースに必要な最低限に絞ったものと言えます。その結果、「既存のプログラミング言語をきれいにまとめなおしたもの」という印象が強いです。

特に、出自が出自なので、J++に近いプログラミング言語でした。つまり、「COM相互運用とデリゲートを持ったJava拡張」から出発しています。

それに加えて、ユーザー定義の値型（struct）、プロパティ、属性、foreachなどの機能が追加されています。主に、実行性能面への配慮と、Javaへの不満の解消となっています。

●開発ツールとの連携

「C#といえばVisual Studio」というくらい、C#は開発ツールとの連携を考えることで生産性を高めています。

図3に示すように、C#はコードを書いた直後から常にコンパイルが行われ、リアルタイムにエラーや警告を知ることができます。また、**図4**に示すように、今カーソルがある場所においてどういうコードが書けるのか、補完候補を出してくれます。

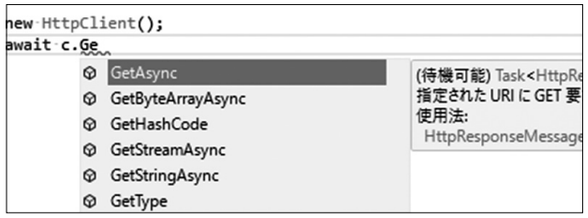
リアルタイムな解析を行うために、C#はコンパイルの速度にも気を使って言語設計・実装されています。また、補完候補が出しやすいような語順で文法が決められていたりします。

また、**図5**に示すように、Visual StudioにはGUIをデザインするためのデザイナーなども付属しています。

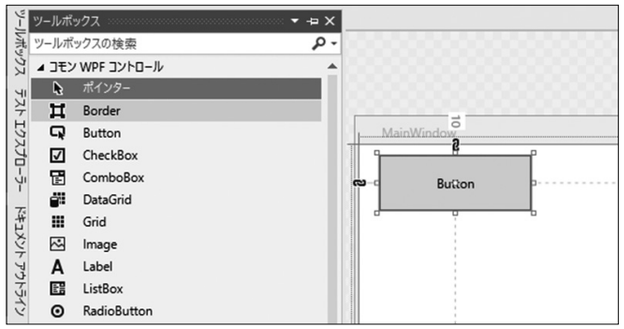
このGUIデザイナーのようなツールがC#コードを自動生成する場合があります（実際、

○**図3　リアルタイムにエラーを検知**

○**図4　コードの補完**



○**図5　GUIのデザイナー**



第1章

Hello, Worldで学ぶC#の基本

シンプルなプログラムで、C#でプログラミングする際に必要なツールや知っておくべき用語、さらにプログラムの全体像まで紹介します。

Note
ご注意

本書のコードは、Visual Studio 2015 RC（Release Candidate）版で作成／検証しています（正式版では、メニュー／操作手順などが変化する可能性もありますので、注意してください）。

まずは、Visual Studio 2015とC#を使って、ごく基本的な「Hello, World」アプリを作成してみます。ごくシンプルなプログラムですが、おさえるべき点は盛りだくさんです。誤解のしようもないコードで、C#の基本的な構文ルールを理解してください。

Visual Studio 2015 RCのインストール

まずは、環境を整えておきましょう。Visual Studio 2015には、上位エディションからEnterprise ／ Professional ／ Communityが用意されていますが、本書では無償で利用できるCommunityエディションを採用します。

Visual Studio Community 2015のインストーラーは、次のページから［Community 2015 RCをダウンロード］ボタンをクリックすることで、ダウンロードできます。

- Visual Studio 2015 RCダウンロード
  https://www.visualstudio.com/downloads/visual-studio-2015-downloads-vs

ダウンロードしたvs_community.exeをダブルクリックすると、**図1**のような画面が表示されます。インストール先のフォルダー／インストールの種類はデフォルトのまま、［インストール］ボタンをクリックします。インストールの種類として「カスタム」を選択した場合には、インストールすべきコンポーネントを細かく選択することもできます。

環境にもよりますが、インストールには数十分～1時間以上かかります。**図2**のような画面が表示されたら、インストールは完了していますので、［今すぐ再起動］ボタンをクリックして、コンピューターを再起動してください。

C#アプリの作成

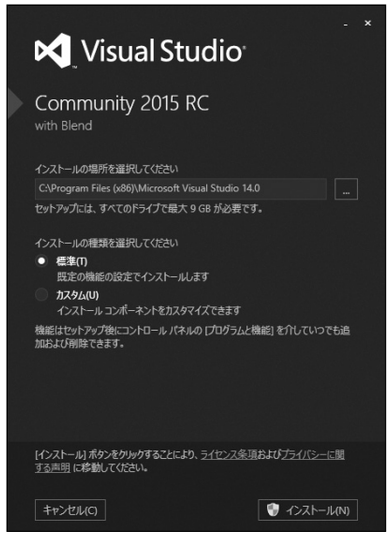
Visual Studioをインストールできたところで、さっそく、アプリを作成していきましょう。入力した名前に応じて、「こんにちは、●○さん！」という挨拶メッセージを表示する、いわゆるHello,Worldアプリです（**図3**）。誤解のしようもないシンプルなコードで、Visual Studioの基本的な操作方法、C#の基本構文などをおさえていきましょう。

●［1］Visual Studioを起動する

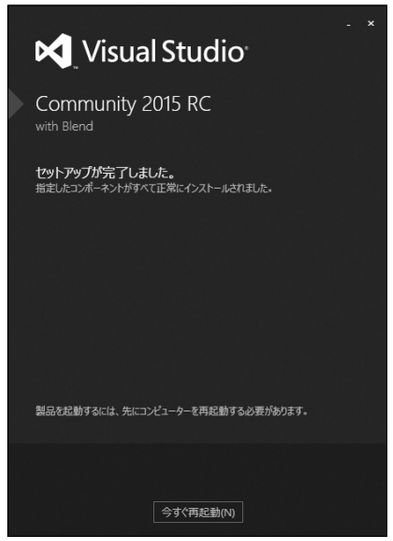
Visual Studioを起動するには、スタート画面から ⬇ ですべてのアプリを表示し、［Visual Studio 2015 RC］を選択します。

管理者権限で起動するならば、アイコンを

○図1　Visual Studio Community 2015のインストーラー



○図2　インストールの終了



○図3　今回作成するアプリの実行画面

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

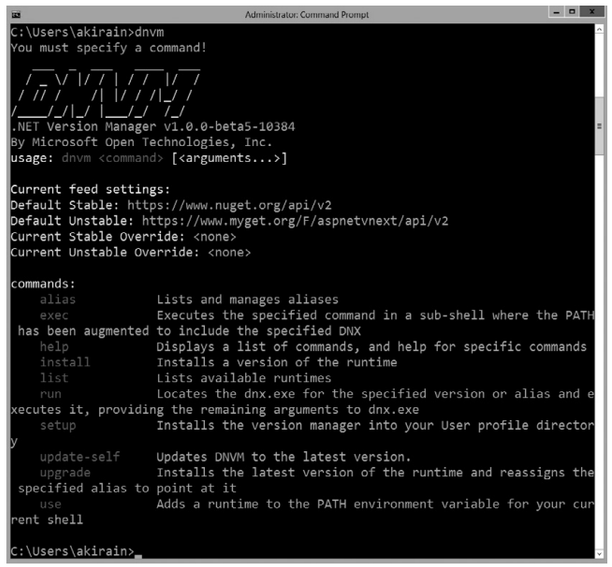
DNVMが正しくセットアップされている場合、コマンドウィンドウでdnvmを入力すると**図4**のように各種情報が表示されます。

その後、dnvmコマンドを実行してDNXをインストールします。

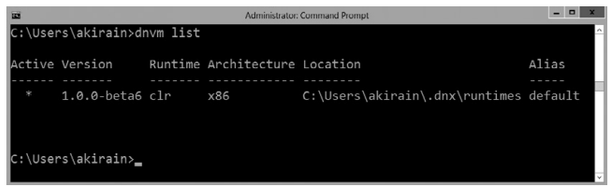
```
> dnvm upgrade
```

次のコマンドで、インストールされたDNXのバージョンなどを確認することができます（**図5**）。

○**図4**



○**図5**




```
> dnvm list
```

Note

DNX環境構築の詳細（Windows）

Windows環境におけるDNX環境構築の詳細は、次のドキュメントなどを参照してください。

・Installing ASP.NET 5 on Windows
http://docs.asp.net/en/latest/getting-started/installing-on-windows.html

●DNXセットアップ（Mac OS X編）

Mono環境のインストール

Mac OS X環境では、現時点ではMonoの実行環境が必要となっています。Monoの環境がセットアップされていない場合には、まず始めにMonoをインストールします。インストールは次の2とおりの方法があります。

・Monoインストーラによるセットアップ

ダウンロードページ（http://www.mono-project.com/download/）より、Mono for Mac OS X（Mono MDK）をダウンロードしてセットアップします。

・Homebrewによるセットアップ

Homebrewパッケージマネージャーを使って、次のコマンドを実行してMonoをインストールすることも可能です。

```
> brew install mono
```

なお、Homebrewパッケージマネージャーのセットアップはhttp://brew.sh/を参照してください。

DNXセットアップ

Mac OS X環境でのDNXセットアップは、ターミナルで次のコマンドを実行します。

```
> curl -sSL https://raw.github
usercontent.com/aspnet/Home/
dev/dnvminstall.sh | sh &&
source ~/.dnx/dnvm/dnvm.sh
```

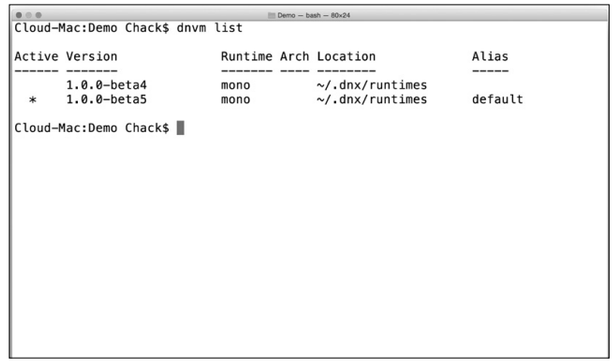
これで、DNVM（.NET Version Manager）がインストールされるので、続いて次のdnvmコマンドを実行してDNXをインストールします。

```
> dnvm upgrade
```

また、次のコマンドで、インストールされたDNXのバージョンなどを確認できます（**図6**）。

```
> dnvm list
```

○**図6**



※Mac OS Xターミナルでのdnvm listコマンド実行画面

Note

DNX環境構築の詳細（OS X）

Mac OS X環境におけるDNX環境構築の詳細は、次のドキュメントなどを参照してください。

・Installing ASP.NET 5 on Mac OS X
http://docs.asp.net/en/latest/getting-started/installing-on-mac.html

●DNXセットアップ（Linux編）

Mono環境のインストール

Mac OS X環境と同じく、Linux環境においても現時点ではMonoの実行環境が必要となっています。

Monoの環境がセットアップされていない場合には、まず始めに次のコマンドを実行してMonoをインストールします。

```
> sudo apt-key adv --keyserver
keyserver.ubuntu.com --recv-keys 3FA
7E0328081BFF6A14DA29AA6A19B38D3D831
EF
echo "deb http://download.mono-
project.com/repo/debian wheezy main"
| sudo tee /etc/apt/sources.list.d/
mono-xamarin.list
> sudo apt-get update
> sudo apt-get install mono-complete
```

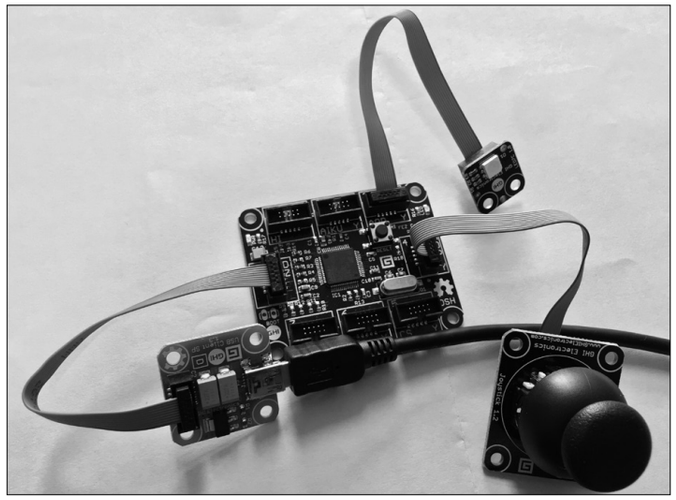
続いて、Linux/Mac OS X上のASP.NET 5アプリケーション実行に使用するKestrelと呼ばれるWebサーバーの実行に必要なCross-Platform非同期IOライブラリのLibuvをインストールします。

```
> sudo apt-get install automake
libtool curl
> curl -sSL https://github.com/
libuv/libuv/archive/v1.4.2.tar.gz |
sudo tar zxfv - -C /usr/local/src
> cd /usr/local/src/libuv-1.4.2
> sudo sh autogen.sh
> sudo ./configure
> sudo make
> sudo make install
> sudo rm -rf /usr/local/src/
libuv-1.4.2 && cd ~/
> sudo ldconfig
```

UIViewControllerやUIWebView、AndroidであればActivityやLinearLayoutなどをC#から直接使用することができ、各プラットフォーム本来のUIを使ったアプリを、ネイティブのパフォーマンスで開発することができます。

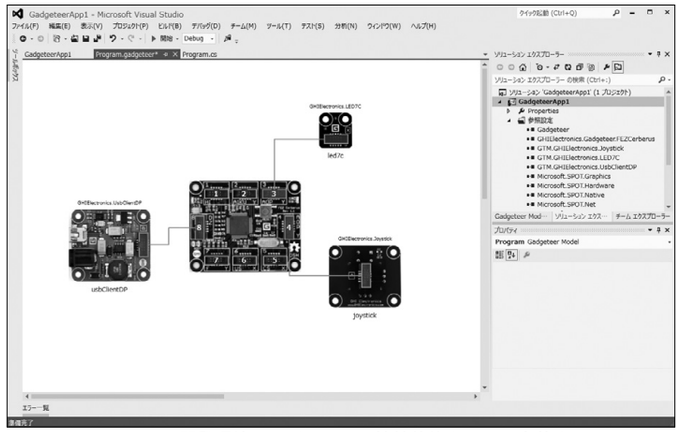
Unityはゲームエンジンであり、ゲームの開発の他、3Dを操作するアプリケーションの場合に使用することもあります。プログラムを実行する部分はXamarinと同じMonoが使用されていて、C#を使ってゲームを作り、Windows、Mac、Android、iOS、Web、ゲーム機（PlayStation3、Xbox 360、Wii）で動作させることができます（**図3**）。すでに多くのゲームがUnityで開発されており、Unity公式サイトのゲームリストのページ（http://japan.unity3d.com/gallery/made-with-unity/game-list）には有名タイトルを含め数多くのゲームがUnityで開発されたとして掲載されています。

○**図4**



※.NET Gadgeteerの一例（FEZ Cerberus）

○**図5**



※.NET Gadgeteerの開発環境。モジュールの接続状態をGUIで定義すると、そのモジュールのクラスのインスタンスがプログラム上に作成されてすぐにコードを書きはじめることができる

C#でのIoT機器開発

IoTの分野では2007年にリリースされた.NET Micro Frameworkとその応用製品である.NET Gadgeteerがあります。

.NET Micro Frameworkは組み込み向けに開発されたOSであり、センサーノードなどの小型機器開発に使用できます。最小構成では256KB ROM、64KB RAMのメモリ構成という、とても小さな環境でC#で記述したプログラムを動作させることができます[注2)]。

.NET Gadgeteerは.NET Micro Frameworkをベースにラピットプロトタイピングをするために作られた環境で、センサー・スイッチなどがモジュールとして用意され、これをメインボードにケーブルで繋ぎ、C#で作ったプログラムでこれらモジュールを制御することができます（**図4、5**）。実製品の開発には向かないものの、簡単にIoTの体験やプロトタイピングができ、入門用として優れています。

.NET Gadgeteerモジュールの仕様は一般公開されており、中に流れる信号はI2CやUARTなど一般的なものであるため、一般の開発者がモジュールを新規に作成することもできるようになっています。.NET Gadgeteer関連の製品を多く製造しているGHI Electronics社のサイトにあるCreations（https://www.ghielectronics.com/community/creations）ではユーザーが独自に開発したモジュールを販売するページも用意されています。

.NET Micro Framework対応のマイコンボードや.NET Gadgeteerの各種部品は、国内ではデバイスドライバーズ社のTinyCLR.jp（http://tinyclr.jp/）で購入できます。

注2）デバイスドライバーズ社の「.NET Micro Framework開発のためのヒント」（http://www.devdrv.co.jp/NETMF/）

Xamarinを用いたスマートフォン開発の実際

話はスマートフォン/タブレットアプリ開発に戻ります。ここからはXamarinを使用してC#を使ったスマートフォンアプリ開発を行う方法を見ていきます。

実際にXamarinを使って開発する際は、Xamarinのライセンスを購入することになります。Xamarinのライセンスは無料のStarter、個人向けのINDIE、法人向けのProfessionalとEnterpriseがあります（**図6**）。

Starterは無料ですが、出力されるプログラムコード（MSIL）のサイズが128KBまでのアプリに制限されています。Indieライセンスは個人と社員が5人までの企業が契約でき、プログラムのサイズの制限が撤廃されます。Professionalになると、Visual Studioを使った開発や、WCFといった機能が使用できます。Enterpriseは1営業日回答、ホットフィックスなどのサポートが利用できる他、Androidの開発においてはILコードに対するAOTを使用できるようになります。

なお、チームでXamarinを使ったクロスプラットフォームの開発を行う際は、必ずしも全員にiOS/Androidのライセンスを割り当てる必要はなく、最終的にビルドするアプリがiOSだけの人、Androidだけの人にはそれぞれを割り当てるのみで問題がありません。割り当ての変更も管理者が自由に行うことができます。

○**図6**

※Xamarinの価格表（https://store.xamarin.com/）。これ以外にもStarterがあり、出力できるプログラムのサイズに制限がある。Visual Studio CommunityではStarterでの開発を行うこともできる

第2章

Gitによるバージョン管理

本章では、近年、バージョン管理システムの標準の１つとなりつつあるGitを例に、バージョン管理システムの基本的な概念と利用方法を紹介します。

バージョン管理システムの基本

アプリケーション開発においてソースコードのバージョン管理システム（VCS：Version Control System）は欠かすことができません。もしバージョン管理システムがなければどうなってしまうでしょうか。筆者はバージョン管理システムなしの開発を経験したことがありますが、非常に苦労したことを鮮明に覚えています。

まず、複数人で同時に1つのファイルを修正することができません。そうすると、Aさんが行った修正によりBさんの修正がなかったことにされてしまうかもしれないからです。Aさんがソースコードを修正している間、Bさんは別のことをして待っている必要があります。

また、過去の修正内容を追跡するのが非常に困難になります。例えば1日に1度ソースコード全体のバックアップを取るなどすれば、1日ごとのソースコードの差分を取ることはできます。しかしそれでは1日ごとにしか差分が取れないし、そもそもどの差分が誰の修正によるものなのかわかりません。

適切にバージョン管理システムを使えば、このような不自由は解消できます。その結果、開発者は本来の開発作業により集中できるようになり、生産性が向上します。

● リポジトリ

バージョン管理の基本は、「いつ」「誰が」「どんな変更を行ったか」を記録することです。これらはリポジトリ（Repository）と呼ばれるデータベースに保存されます。データベースといっても、Gitを含めほとんどのバージョン管理システムでは、リポジトリはファイルです。バージョン管理システムのユーザは、各種コマンドを利用してリポジトリに書き込んだり参照したりします。

● コミット

リポジトリに記録を追加するのがコミット（Commit）です。開発者がコミットを行うと、リポジトリに「いつ（タイムスタンプ）」「誰が（開発者）」「どんな変更を行ったか（前回のコミットからの修正内容）」が記録されます。

では、開発者はいつコミットすれば良いのでしょうか。プロジェクトによっては「いつすべきか」「いつしてはならないか」が明確に定められている場合もあるでしょうが、コミットのコマンドを実行するのは開発者ですから、基本的には開発者次第です。

ただ一般的には、数分から数時間に1回はコミットし、1日以上コミットしない状況は避けるべきとされています。理由はいくつかありますが、例えばソースコードレビューのレビュアーのためです。1コミットあたりの差分が小さければ、修正の目的が何なのか、比較的容易に把握できます。逆に1コミットでの修正量が多いと、どの箇所の修正が何を目的としているのか把握するのが困難になってしまいます。他にも、万が一開発マシンが壊れた時に失われる情報をできるだけ少なくするなどの理由があります。

コミット先のリポジトリの場所は、バージョン管理システムによって異なります。後述しますが、Gitでは開発者のローカルマシンにあるローカルリポジトリがコミット先です。

● ブランチ

複数のチームで1つのプロダクトを開発しているケースを考えてみましょう。チームAでは機能1を開発していて、チームBでは機能2を開発しています。ある日、チームAが機能1の開発を終えたとします。すぐにでも機能1をリリースしたいですね。しかし、チームBによる機能2の開発が、まだ完了していません。チームAとBはリポジトリを共有していますから、作りかけの機能2のためのソースコード修正がリポジトリに含まれてしまっています。これでは機能1をリリースできません。

このような課題を解決するのがブランチ（Branch）です。ブランチとは直訳すると「分岐」のことで、履歴の流れを分岐して記録していく仕組みです。分岐したブランチは、他のブランチに対する修正の影響を受けません。先ほどの例で言えば、機能1を開発するためのブランチ1と、機能2を開発するためのブランチ2を作れば良いことになります。そうすれば、ブランチ1には機能1のための修正しか含まれませんから、完成していないブランチ2の状況に影響を受けることなく、リリースすることができます。

Git

Gitは、現在最も利用されている分散型のバージョン管理システムです。Linuxの開発者であるLinus Torvaldsにより、Linuxのソースコードを管理するために開発されました。その後オープンソースとして公開され、多くのプロダクトで利用されています。現在、Gitのホスティングサービスである GitHubとともに、広く普及しています。

● 中央集権型と分散型

バージョン管理システムは、大きく中央集権型と分散型に分類できます。中央集権型と分散型の最大の違いは、セントラルリポジトリの有無です。セントラルリポジトリは唯一無二のリポジトリで、すべての履歴を集中管理します。すべてのユーザがそこに対してコミットします。

セントラルリポジトリがあるのが中央集権型のバージョン管理システムです。逆に分散型のバージョン管理システムは、セントラルリポジトリを持ちません。代わりに各開発者のマシンにリポジトリを持ちます。それらはセントラルリポジトリに対してローカルリポジトリと呼ばれます。

中央集権型は構造がシンプルです。しかし、セントラルリポジトリにアクセスできる環境