SECTION 02

GitHub Desktopをインストールする

GitHubが無料で公開しているGitHub Desktopをインストールしましょう。GitHubと連携して使うことが多いため、先にGitHubのアカウント（ユーザー名）を作成しておきます。GitHubのユーザー名はGitでバージョン管理するときに「誰が変更したか」を表すために使われるので、長く使う名前ということを考えて決めてください。

◎ GitHubアカウントを作成する

GitHub Desktopをインストールする前に、GitHubのユーザー名とパスワードを決めてアカウント（利用権）を作成します。GitHubのユーザー名は、GitHubのサービス上だけでなく、**Gitでファイルを更新したユーザーを表す名前**としても使われます。長く使うものですから、自分だけでなくほかの人が見ても誰のことかがわかる名前にしましょう。GitHubをよく使う人の中には、TwitterなどのSNSのユーザー名と統一していることもあります。「unknown1234」のような適当な名前でも登録できますが、あまりおすすめしません。

POINT

Gitのユーザー名とGitHubのユーザー名は別にすることもできます。ただし、それだとややこしいので合わせることをおすすめします。

まずは、Webブラウザーで GitHubのサイトを表示します。GitHubのサイトは、英語のgithub.comと日本語のgithub.co.jpがありますが、日本語サイトでも途中から英語サイトに飛ばされるので、ここでは最初から英語サイトで登録します。

- **GitHub（英語サイト）**
  https://github.com/

SECTION 03

Visual Studio Codeをインストールする

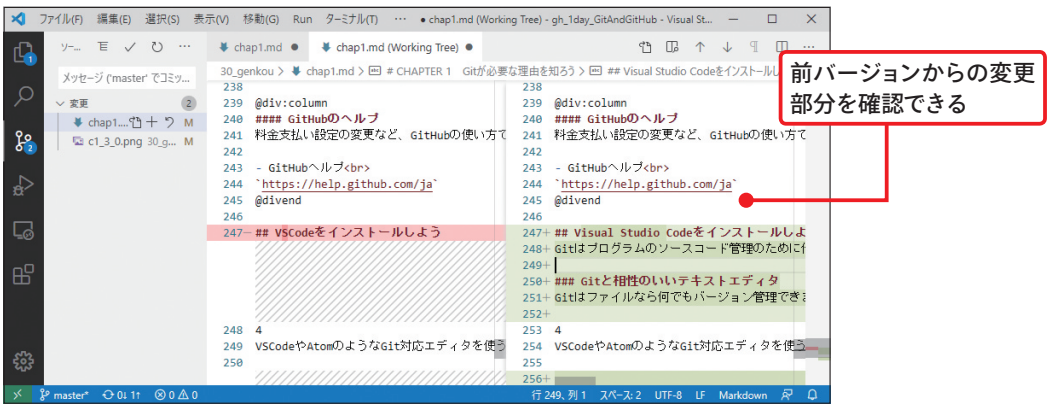
Gitはプログラムのソースコード管理のために作られたため、ソースコード編集用のテキストエディタやIDEと組み合わせて使われることがよくあります。ここではGitと連携する機能を持つVisual Studio Code（以降VSCode）のインストール方法を解説します。

◎ Gitと相性のいいテキストエディタ

Gitはファイルなら何でもバージョン管理できますが、最大の用途はプログラムやWebページなどのソースコードファイル（プログラミング言語で書かれたテキストファイル）の管理です。そのため、ソースコードの編集に使用するテキストエディタやIDE（統合開発環境）の中にはGitと連携する機能を持つものがあり、プログラム開発以外でも役立ちます。本書ではその中からマイクロソフト社のVSCodeを紹介します。

VSCodeのソース管理画面では、前回のバージョンからどこが変更されたか（差分）をわかりやすい形で表示できます。また、コンフリクトの解消や、前バージョンまで戻すといったGitの一部の機能を利用できます。

図1-16 VSCodeのソース管理画面



SECTION 04

Gitの基本的なしくみを理解する

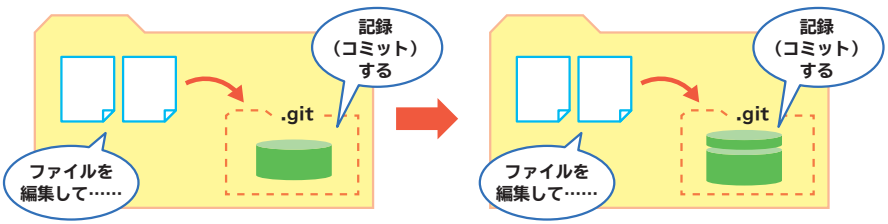
GitHub Desktopの操作は比較的シンプルなのですが、Gitのしくみや用語がわかっていないと意味がまったく理解できません。逆にしくみさえわかっていれば、操作は簡単に覚えられます。そこで次章から実際の操作の説明を始める前に、Gitの基本的なしくみと用語を解説します。

◎ ただのフォルダを「リポジトリ化」する

Gitを利用するには、パソコン内のフォルダを、保管庫を意味するリポジトリに変えます。リポジトリ化すると、フォルダ内に.gitという名前の隠しフォルダ（通常の方法では見られないフォルダ）が作られます。この.gitフォルダがリポジトリの本体です。

あとはファイルを編集してはリポジトリ（.gitフォルダ）に記録、編集しては記録を繰り返していきます。このリポジトリに記録する操作をコミットといい、記録したひとかたまりのデータのこともコミットと呼びます。

図1-30 リポジトリ（.gitフォルダ）の基本的な使い方



リポジトリ化したフォルダの中は2つの世界に分かれます。1つはGit以外のプログラムから見える通常の世界です。これをワーキングディレクトリと呼びます（ワークツリーまたは作業ツリーと呼ばれることもあります）。一般的なオフィスソフトやグラフィックスソフト、エディタなどで操作できるの

SECTION 01

ローカルリポジトリを作成する

この章ではGitHub DesktopとVSCodeを使ったローカルリポジトリの使い方を学びます。まずはローカルリポジトリを作ってみましょう。ローカルリポジトリを作る方法はいくつかありますが、ここでは何もない状態から作成する方法を説明します。

◎ ローカルリポジトリを作成する

ローカルリポジトリの作成方法は3つある

GitHub Desktopを使ってパソコンの中にローカルリポジトリを作りましょう。ローカルリポジトリの作り方には次の3通りがあります。

❶ハードディスク内に新規ローカルリポジトリを作成

ローカルリポジトリもリモートリポジトリもない状態からスタートする場合は、＜File＞→＜New repository＞を選択して作成します。Gitを使わずに作業を進めていて、新たにGitを利用することにした場合も、この方法でリポジトリ化します。

❷リモートリポジトリをコピーしてローカルリポジトリを作成

すでにリモートリポジトリがある場合は、＜File＞→＜Clone repository＞を選択するか、Webブラウザーで GitHub上のリモートリポジトリを表示して＜Open in Desktop＞を選択します（P.76参照）。リモートリポジトリをローカルにコピーすることを「クローンする」または「クローンを作成する」といいます。

❸すでに作成済みのローカルリポジトリを取り込む

他のGitクライアントで作成したローカルリポジトリをGitHub Desktopで利用できるようにするには、＜File＞→＜Add local repository＞を選択します。

SECTION 02

変更を「コミット」する

新しいファイルを作成して、コミットしてみましょう。ファイルを新規作成したり編集して保存したりしても、コミットするまではリポジトリには何の影響もありません。ここではファイルの編集／保存とコミットの関係をつかんでください。

◎ ファイルを新規作成する

VSCodeでファイルを作成し、変更して保存するところまでやってみましょう。**Markdown**という形式のテキストファイルを作ります。ファイルの拡張子は「md」としてください。Markdownについてはあとのセクションで解説するので、とりあえず本の通りに操作してください。

図2-7 新規ファイルを作成

SECTION 04

Markdownの書き方を覚える

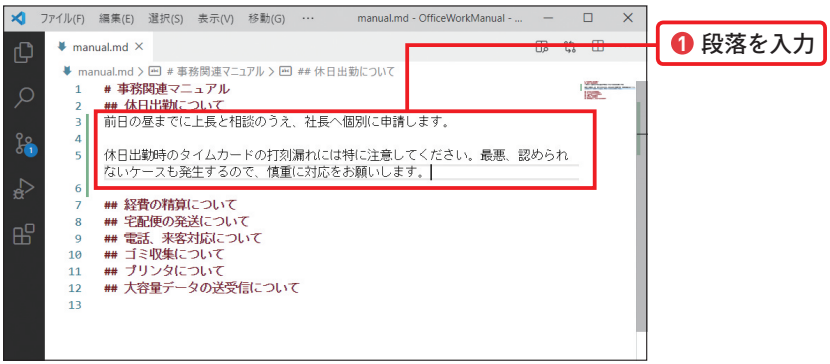
Markdownはテキストファイルの記法の一種で、いくつかの記号を使って見出しや箇条書き、表などの簡単な書式を設定できます。そのまま読むこともできますし、HTMLに簡単に変換することもできます。GitHub向けのドキュメントではMarkdownが使われるため、書き方を覚えておきましょう。

◎ 見出しと段落を入力する

Markdownはテキストファイルの一種ですが、その記法によって「見出し」「箇条書き」「強調」「表」「画像」などを表現できます。つまりMicrosoft Wordのような書式の表現が可能なのですが、Wordのファイル（docx）がバイナリファイルなのに対し、Markdownは拡張子が「md」のテキストファイルです。そのため、Gitでの管理に適しています。また、Markdownは容易にHTMLに変換することができ、Webで利用するドキュメントの記述に使われています。GitHubでもよく使われているので、その基本的な書き方を覚えておきましょう。

まず一番シンプルなルールは、行頭に#を付けると「見出し」になり、何も付けないと「段落」になるというものです。VSCodeの見出しのあとにいくつか文章を入力してください。

図2-26 段落を追加



SECTION 03

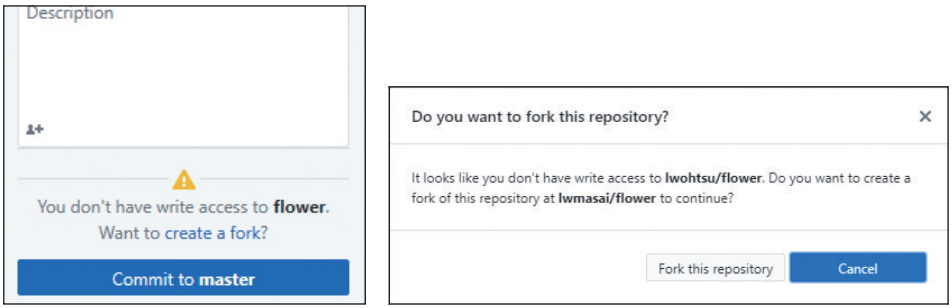
共同作業でファイルを編集する

リモートリポジトリの所有者以外の人がプッシュできるようにするには、コラボレーター（共同編集者）にしてもらう必要があります。コラボレーターの設定を行い、他のユーザーがコミットするとどうなるかを確認しましょう。

◎ コラボレーターを設定する

パブリックのリポジトリは、誰でもクローンでき、プルすることができます。ただし、プッシュはできません。わかりやすくいえば、**読むことはできても書き込むことはできません**。コラボレーターになっていない状態で、コミットをプッシュしようとすると「Want to create a fork?（フォークを作成したいですか？）」「Do you want to fork this repository?（このリポジトリをフォークしたいですか？）」と表示されます。**フォーク（fork）**とは、他人のリモートリポジトリを複製して自分のリモートリポジトリを作ることで、誰かが作ったアプリを改良したアプリを作りたいときなどに利用します。今回は1つのリモートリポジトリを共同編集したいので、フォークでは困ります。

図3-24 フォークを確認するメッセージ



1つのリモートリポジトリを複数人で編集したい場合は、**コラボレーター（共同編集者）**の設定を行います。これは公開も非公開のリポジトリもまったく同じです。

SECTION 02

コンフリクトを解決する

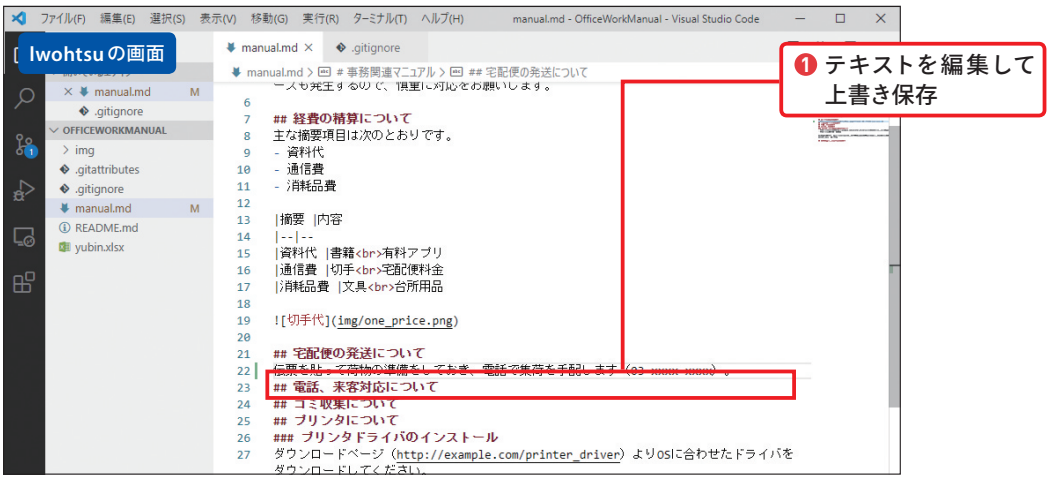
ここではコンフリクトをわざと引き起こし、それを解決してみます。作業者が2人いて同時に変更した場合、あとからプル／プッシュしたほうの画面にコンフリクトの警告が表示されます。警告が表示された側で、矛盾を解消するための編集を行います。

◎ コンフリクトをわざと引き起こす

まずコンフリクトを引き起こしてみましょう。コンフリクトが確実に起きるのは、2人以上が同じファイルの同じ行を変更したときです。

今回は「宅配便」の説明を1人目（lwohtsu）は「電話で集荷を手配する」と書いたことにします。

図4-8 テキストを編集する



編集が終わったので、ファイルを上書き保存し、コミットしてプッシュします。まだ相手がプッシュする前なので、問題なくプッシュは成功します。

SECTION 04

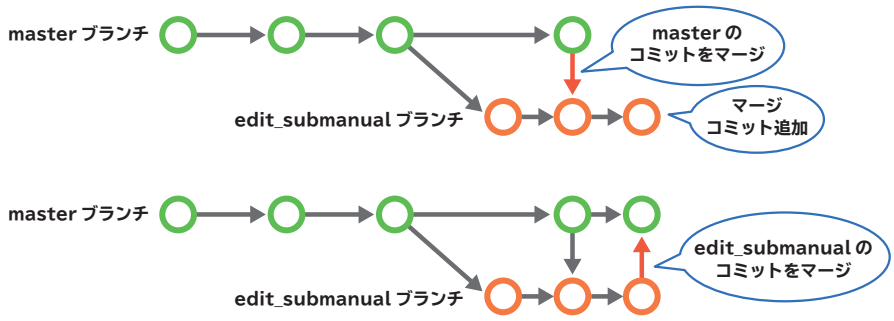
ブランチをマージする

ブランチのマージとは、他のブランチの変更を取り込むことです。作業が完全に終わったあとに行うマージ以外に、作業中の状態を合わせるために行うマージもあります。ここでは両方のマージを行ってみましょう。

◎ さまざまなマージの使い方

ブランチのマージと聞くと、分家のブランチを本家のブランチ（masterブランチ）に統合するというイメージがあります。しかし、本家のブランチから分家のブランチにマージすることもあります。ブランチを分けたまま作業を続けていると差が大きくなってしまうため、時々本家のコミットを取り込んで状態を合わせるのです。

図4-26 2方向のマージ



今回はmasterブランチでmanual.mdが変更されたので、それをedit_submanualブランチに取り込むことにします。この例だとあまり意味はありませんが、sub_manual.mdを編集するときにmanual.mdを参考にするといったシナリオが考えられます。

SECTION 03

プルリクエストを使ってマージする

プルリクエストは、ブランチ上で行った編集結果を、共同作業者に確認してもらってからマージする機能です。掲示板に似た対話型のインターフェースを持ち、変更点を見やすく表示する機能や、確認結果をわかりやすく伝える機能などで構成されています。

◎ プルリクエストとは

第4章ではブランチを使った共同作業を解説しましたが、実際にやってみると「いつブランチを作ればいいのか」「マージした結果に問題があったらどうするのか」などさまざまな問題に突き当たります。マージに伴う問題の解決策として作られたのが、GitHubの**プルリクエスト**です。便利な機能なので、GitHub以外のGitホスティングサービスでも同様の機能がたいてい用意されています。

プルリクエストの使い方として、**GitHubフロー**というものが提唱されています。大まかには、次の2つのルールで作業を進めるというものです。

- **何かの作業をスタートする場合は、必ずmasterブランチからブランチを分岐して進める**
- **ブランチをマージしたいときはプルリクエストを利用し、共同作業者に評価（レビュー）してもらってからマージする**

これなら「いつブランチを作るか」という疑問が解決し、「ほかの人の確認なしにマージして問題が起きる」こともなくなります。

プルリクエストのページにはいろいろな情報が盛り込まれているので、最初は少しとまどうかもしれません。基本的には時系列順のタイムラインになっており、**「ブランチに対するコミット」**と**「共同作業者の指摘（レビュー）」**が並んでいます。SNSのように互いにコメントを投稿して相談を進め、修正が必要ならローカルリポジトリ上で作業してプッシュすると考えるとわかりやすいかもしれません。

最終的にマージして問題ないということになれば、このページ上からマージを行い、プルリクエストをクローズします。

SECTION 04

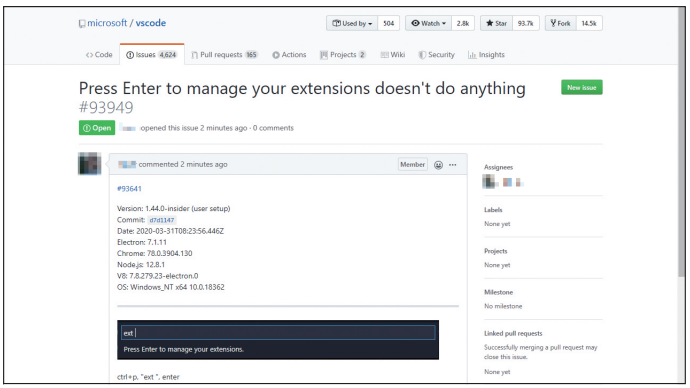
イシューを使って問題を解決する

イシュー（Issue）はアプリのバグなどを報告するために使われる、掲示板に似た機能です。原則的に1つの問題につき1つのイシューを作り、解決したらクローズするという流れで使っていきます。掲示板に似ているとはいえ、本来は課題解決ツールの一種なので、雑談に近い話し合いをしたいときは別にチャットツールなどを併用します。

◎ イシューは問題解決ツール

イシュー（Issue）とは「問題」「課題」などを意味する英語で、GitHubでは、リポジトリで開発しているアプリのバグ報告や、作業場の懸念点を相談するための機能を指します。公開リポジトリでは、コラボレーター以外の誰でもイシューを作成できるので、オープンソースのアプリではバグ報告や追加機能の要請などに使われています。

図5-33 VSCodeのバグ報告イシュー



イシューは問題を見つけた人が作成し、そこに他の人がコメントをしていきます。問題によってはソースコードを修正して解決する必要も出てくるので、その場合はプルリクエストを作成して修正を行いま

SECTION 01

Gitはコマンドラインでも利用できる

ここまでGitHub Desktopを使って解説してきましたが、開発の現場ではコマンドラインのGitもよく使われています。この章ではGitHub Desktopの機能と比較しながら、コマンドラインのGitの使い方を解説します。

◎ gitコマンドとGit Bash

以前からGitを利用している人には当たり前のことなのですが、Gitはコマンドラインでも利用できます。コマンドラインの命令はテキストなので、コマンドを自動実行するツールと組み合わせて自動化しやすいといったメリットがあります。その反面、**コミット履歴やブランチなどの状態を脳内でイメージする**必要があるため、GitHub DesktopのようなGUIツールに比べると少々慣れが必要です。コマンドラインとGitHub Desktopを併用することもできるので、慣れるまではGitHub Desktopで状況を確認しながらコマンドラインで操作してもいいでしょう。

本書では、GitをWindowsで実行するために**Git Bash**を利用します。Git BashはWindows上でLinuxコマンドを利用可能にするコマンドラインツールです。Windows標準のコマンドラインツール（コマンドプロンプト、PowerShell）はLinuxとはコマンド体系が異なるのですが、Git Bashを使えばLinuxやmacOSと同様の操作が行えます。

コマンドラインのGitでは、次に示すような形式の**gitコマンド**を入力して操作します。gitコマンドとサブコマンド、オプションの間は半角スペースを空けてください。

リスト6-1 gitコマンドの体系

```
git サブコマンド オプション／パラメータ
git add -m "コミットメッセージ"
```

POINT

以降はGit Bashの画面での解説となりますが、macOSではターミナルを利用してください。