ソーシャルコーディングの可能性
- Git / GitHub による文書管理を中心に
Potential of "Social Coding" with a focus on Git / GitHub

井　上　貢　一
Koichi INOUE

1. はじめに

我々の住む世界は文書に満ち溢れている。それは音声と文字という２つの側面をもつ「言葉」のアーカイブであると同時に、我々に様々な情報をもたらし、かつ我々の生活を規定する。

法律、定款、企画書・議事録、マニュアル、書籍、Web サイト、そして電子機器をコントロールするプログラムなど。文字データで綴られたものはすべて文書である。

文書は、まずいくつかの言葉が集まることからはじまり、それらが切りつながれて、構造化する。すなわち「編集」によって安定的な形に収斂する。しかし多くの場合、それが生産完成品となることはなく、事後に何らかの修正が施され、更新されることになる。そのままの形で保管される歴史的資料を除いて、法の条文、書籍、ソフトウエアのプログラムも、すべてにバージョン履歴が存在するもので、我々は常にそれらを最新の状態に保ち、関連する知識を更新しなければならない。

さて、この文書の編集・更新という作業は、個人の日記のようなものを除いて、複数の関係者によって行われるのが一般的である。そして今日、その作業の多くは、インターネットの存在を前提にクラウド上で行われるようになった。

世界中の有志によって今も更新されつづけている LinuxOS がその典型であるが、様々なネットワークサービスの登場によって、文書の管理手法は大きく変わりつつある。さらにそれは知的財産というものに対する人々の意識をも一変させた。

本稿では、その潮流に新たな変革をもたらした Git と GitHub を中心に、文書管理と共同編集に関する全体像を俯瞰し、そのキーワードとなったソーシャルコーディングの可能性を考察する。

2. 文書の管理に関わるツールとサービス

文書は通常、紙をバインダーで綴じるか、デジタルファイルとしてフォルダにまとめるなどの方法で管理されている。デジタルデータの場合、その多くはファイル名に日付を付けるなどの方法で、初稿から最終稿まで順次保管されている。

しかし、共同編集によって次々に状態が更新されると、「どれが最新のものかわからない」、「どこが修正されたのかわからない」といった問題が発生する。また複数の編集者が同時に同じ箇所に異なる修正を行った場合 (衝突が生じた場合)、プロジェクトはそこで一旦フリーズしてしまう。

ソフトウエア開発の現場では、こうした問題に対処するため、古くからバージョン管理システムというものが存在した。

かつてそれはキーボードでコマンド入力する CUI(Character User Interface) が主流で、専門家以外には縁遠い存在であったが、近年一般的なパソコンの操作スキルがあればこれを簡単に利用できる GUI(Graphical User Interface) ベースのツールが登場したことで、バージョン管理というものが非常に身近になった。

また、このバージョン管理システムに対応したホスティングサービスも増えて、メースアドレスを用いてアカウントを取得すれば、誰もが自由に利用できるようになったことで、プログラム開発、Web サイト制作、さらには雑誌の編集や、公文書の公開まで、インターネットを介した共同編集が気軽に行える環境が整った。

以下の節では、2.1. バージョン管理システム、2.2.GUI クライアント、そして 2.3.OSS ホスティングサービスについて、その現状と動向に関して概説したい。

2.1. バージョン管理システム

我々の日常業務では、ファイル更新の際、古いファイルのファイル名に年月日などを付けて変更したり、新規に作成するファイルに年月日などを付けて作成するなど、ファイルの版数 (バージョン) の管理を行うのが普通である。ワープロ (Word) やスプレッドシート (Excel) であれば、作成者や変更履歴を記録することもできるが、現在業務で扱うデータは多岐に渡っており、これらのデータを全てバージョン管理するためには、管理者を置くか、組織内ルールをつくるなど、多くの手間と時間が必要になる。そこで登場したのがバージョン管理システムである。

バージョン管理システムは、ファイルの作成者、作成日付、更新日付、更新者、変更履歴、コメントなど、ファイルに関する 5W1H を「リポジトリ」と呼ばれる場所に保持し、複数のメンバーのアクセスに対応して、その変更履歴を記録する。

それには様々な製品があるが、大きく以下の３種類に分類される。

1) 単独型

ローカルマシン上のファイルのバージョンを管理するタイプ。リポジトリに他人がアクセスすることのない自分専用のリポジトリといえる。

2) 集中型 (クライアントサーバー型)

サーバー上のリポジトリを複数の利用者が遠隔操作して、バージョンを管理するタイプ。クライアントサーバー間の接続が前提となる。

3) 分散型

サーバー上のリポジトリを複数の利用者がローカルに複製、ローカル上での編集結果をリモートリポジトリに反映させるタイプ。リポジトリの複製をクライアント側で保持するため、ネットワークに接続できないオフライン状態でも、リポジトリの操作を行うことができる。

Git　Mercurial　Subversion

図 1. 様々なバージョン管理システム

図 1 はキーワード「バージョン管理システム」で検索した際に上位にヒットした Git(分散型)、Mercurial(分散型)、Subversion(集中型)、以上３つの公式サイトである。

1) Git(ギット)

2005 年に登場した分散型のバージョン管理システム。Linux の開発者リーナス・トーバルズ (Linus Torvalds 1969-) らが共同開発したもので、LinuxOS の開発管理にも使用されている。

2) Mercurial(マーキュリアル)

2005 年に登場した分散型システムで、Mozilla Firefox の開発管理などに使用されている。

3) Apache Subversion(SVN)

2000 年に登場した集中型システムで、それまで中心的存在であった CVS(Concurrent Versions System) を改善する目的で開発されている。

これら３つのシステムは、いずれも現役でシステム自体の開発が継続されているものであるが、

図 2. バージョン管理のトレンド

GoogleTrends[*1] で過去 5 年間の検索キーワードのトレンドを比較すると、Git への関心に高まりがあることがわかる (図 2)。

図 3 は Git に代表される分散型バージョン管理の具体的なイメージであるが、まずは重要なキーワードについて、以下に概説する。

1) ローカルリポジトリ (Local Repository)

プロジェクトに関わる個々のメンバーが、自分のマシン上に用意する「データの管理スペース」。

2) リモートリポジトリ (Remote Repository)

遠隔サーバ上にあって、複数のメンバーが共同編集するためのスペース (中央リポジトリ)。

3) インデックス (Index)

リポジトリに変更登録 (Commit) する内容を事前に追加 (Add) する場所。集中的な編集作業の後、全体一括、あるいは逆に、必要な部分のみを選択するために設けられた中間ステージ。

4) ワークフォルダ (Work Folder / Work Tree)
　バージョン管理システムの管理下に置かれた作業ファイルが存在するフォルダ。作業ツリー (ワークツリー) ともいう。
5) コミット (Commit)
　ファイル・フォルダの追加や変更を、ローカルリポジトリへ登録する操作。
6) プッシュ (Push)
　ローカルリポジトリの変更点を、リモートリポジトリに送って反映させる操作。
7) プル (Pull)
　リモートリポジトリから変更点をダウンロードしてローカルリポジトリに反映させる操作。
8) クローン (Clone)
　リモートのリポジトリを複製して、ローカルにリポジトリを作成する操作。既存のプロジェクトに新規参入する場合に、参加者が最初に行う操作である。
　集中型のシステムでは、リモートサーバーに置かれた中央リポジトリのみでファイルが管理されるのに対し、分散型のシステムでは、メンバーが個々の PC に個別のローカルリポジトリを作成し、これを更新する。ローカルリポジトリに対しては、いつでも変更の登録 (コミット) が可能で、オフラインでも作業が可能なこと、また、複数のリポジトリが作成されることで、破損事故における復旧も比較的容易にできるなどのメリットがある。

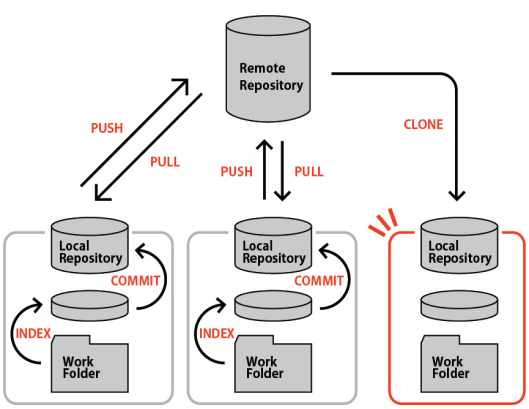


図 3. 分散型バージョン管理のイメージ

2.2. GUI クライアント

　バージョン管理システムを使うには従来の CUI でも可能だが、現在では GUI の利用が主流となっている。特に Git を GUI で扱うことのできる Git クライアントには、Source Tree、git GUI、smart Git、GitHub Desktop、Tower、tortoisegit など様々なものが存在する。図 4 は「Git クライアント」で検索した際に上位にヒットした GUI ツールの公式サイトである。

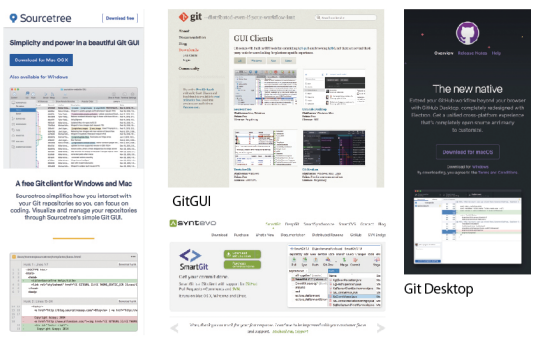


図 4. 様々な Git クライアント

1) SourceTree
　SourceTree は Atlassian 社による Git クライアントで、Windows、Mac いずれの OS にも対応している。インターフェイスは日本語に完全対応しており、ウインドウの構成も直感的である。
2) git GUI
　Git の公式 GUI である。メモリ占有もわずかで、動作は軽快であるが、英語ベースで日本語化に難があるため、日本語の情報は少ない。
3) smart Git
　Windows や Mac 対応の GUI クライアントは多いが、もともと CUI に慣れたユーザーの多い Linux(Ubuntu) に対応したものは少なく、Ubuntu ユーザにとっては、貴重な候補となる。


図 5. Git クライアントのトレンド

　この 3 者について GoogleTrends で過去 5 年間の動向を見ると、Source Tree に関心が集中していることがわかる (図 5)。

2.3. OSS ホスティングサービス

OSS ホスティングサービスとは、オープンソースのソフトウェアを格納するリポジトリを中心に、バージョン管理システム、ホスティングサーバー、開発者同士のコミュニケーションツールをインターネット上で提供するサービスである。

Wikipedia の記事「OSS ホスティングサービスの比較」では世界中に 50 以上のサービスがあり、そのうち Git に対応したものが 15 件ほどある。

図 6 は、「OSS ホスティングサービス」で検索し、上位にヒットした公式サイトである。

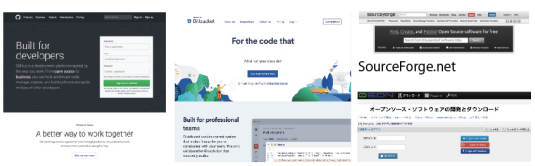


図 6. 様々な OSS ホスティングサービス

以下、サービスが開始された年代順に各サービスの概要を紹介する。

1) SourceForge.net

協働型バージョン管理・ソフトウェア開発管理システムで、オープンソースのソフトウェア開発においては管理と制御の中心的存在である。1999 年に設立され、現在は Geeknet 社が運営している。提供しているバージョン管理システムは CVS、SVN、Git、Mercurial、Bazaar である。

2) OSDN(Open Source Development Network)

SourceForge.net の日本語版サイトとして、VA Linux Systems Japan の OSDN 事業部によって 2002 年に設立。日本のオープンソースソフトウェアプロジェクトに特化したホスティングサイトで、現在は VA Linux からスピンオフした OSDN 株式会社によって運営されている。バージョン管理システムは SourceForge.net と同様に、CVS、SVN、Git、Mercurial、Bazaar で、その他に、プロジェクト Wiki、プロジェクト Web、また Sourceforge.net にはないコンパイルファーム (ソフトウエアをビルドするためのサーバー群) も提供している。

3) GitHub

GitHub, Inc. が運営するソフトウェア開発プロジェクトのための共有ウェブサービスである。2008 年に設立され、現在では最もポピュラーな Git ホスティングサイトといわれる。このサービスは SNS 機能も持っており、開発者は自身のバージョンのリポジトリをネットワークグラフによって視覚的に把握できるようになっている。

オープンソースの開発を前提としたパブリック (公開) 利用の場合、アカウントは無料で取得可能。ビジネスユーザー向け、教育機関向け、個人 (非公開) 用途など、様々な有料プランも展開している。

4) Bitbucket

2008 年、独立ベンチャーによって設立され、現在は Atlassian 社が運営するソフトウエア開発のための Web ベースのホスティングサービスである。GitHub と異なるのは、バージョン管理システムとして Git と Mercurial のいずれも利用できる点、また無料のアカウントでもプライベート (非公開) リポジトリを作成できて、5 ユーザーまでの小規模チームであれば、非公開リポジトリの数は無制限につくることができる点である (2017 年 10 月現在)。

これは開発メンバーの規模が小さい Web サイトの制作には適した環境で、Web 系の様々な記事で Bitbucket の活用法が紹介されている。

図 7. ホスティングサービスのトレンド

GoogleTrends で 過去 5 年間の動向を見ると、GitHub に関心が集中していることがわかる (図 7)。

関連する記事 [*2] によれば、プログラミングとコミュニケーションを融合させたことが、GitHub の人気を高めたようだ。

GitHub のスローガンは「ソーシャルコーディング (Social Coding)」である。ネットワーク上で様々なコラボレーションを展開する若い世代にとっても、オープンソースコミュニティの文化は共感されやすい。

3. Git と Git クライアントの利用

本節では、Git と SourceTree を例として、具体的なバージョン管理の導入手順を概説する。

3.1. SourceTree のインストール

まずは環境の準備から。SourceTree をインストールすると同時に Git を内臓した環境が構築できる。ダウンロードの公式サイトは以下。

https://www.sourcetreeapp.com/

3.2. アカウント登録

SourceTree では、リポジトリの管理者を特定するため、システムの環境設定で、ユーザ名とメールアドレスを登録するのが一般的である。また GitHub などのホスティングサービスと連携する場合には、そのアカウント情報も登録する。

3.3. ローカルリポジトリの作成

はじめに、個人のファイル管理を前提に概説する (共同編集は 4 節で後述)。SourceTree のメニューから「新規>ローカルリポジトリを追加」で、任意のフォルダにリポジトリを作成できる。

図 8 に示すとおり、GUI でのフォルダ閲覧では、作業ファイルが見えるだけだが、ターミナルで隠しファイルを確認すると .git というディレクトリがあって Git の設定ファイルや履歴が管理されていることが確認できる。

図 8. ローカルリポジトリ

3.4. 編集 / インデックス / コミット

SourceTree におけるバージョン管理は、以下のような手順で進行する。

1) フォルダ (SourceTree では作業ツリーあるいはワークツリーという) にあるファイルをエディタで編集・更新すると、SourceTree はその変更を検知し、Uncommitted Changes と表示する。

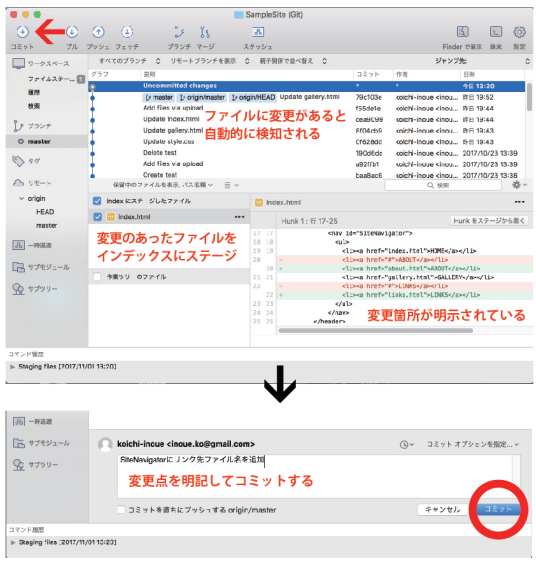


図 9. ファイルの変更からコミットまでのプロセス

2) 変更を登録すべきタイミング、すなわち、以前の状態と現在の状態をバージョンを区別しておきたいタイミングで、作業ツリーのファイルにチェックする。すると、当該ファイルがインデックスにステージ (Add) され、コミット (Commit) すなわち変更の登録が可能な状態になる。

3) ウインドウ左上の「コミット」をクリックし、変更点に関する説明 (これは必須の情報) を記載して、右下の「コミット」を実行する (図 9)。

以上で、バージョン変更が記録され、ウインドウ上では履歴が更新されるとともに、変更点に関する説明、コミット ID(ランダム文字列)、作者、日時の情報が表示される。

3.5 ブランチ (Branch)

Git では作業履歴を分岐することもできる。主幹となる統合ブランチ (master) に対し、機能追加やバグ修正といった作業を行うトピックブランチ (develop) を分岐させ、最終的に統合ブランチにマージするといった工程が可能で (図 10)、共同編集には必須の機能となっている。

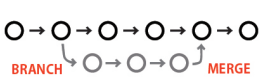


図 10. ブランチとマージのイメージ

4. GitHub の利用

共同編集にはリモートリポジトリを置けるホスティングサービスが必要である。本節では GitHub の利用を前提にその手順を概説する。

4.1. アカウントの取得

GitHub を利用するには、まずアカウントの取得が必要である。手続きは簡単でメールアドレスの認証が済めばすぐに利用できる。個人 User アカウントからスタートすることになるが、さらに Organization アカウントの追加が可能で、ここに他の GitHub ユーザーを招待することで、リポジトリの共同編集が可能になる。

4.2. 中央リポジトリの構築

まず GitHub すなわちリモートサーバー上に中央 (リモート) リポジトリが必要となる。一般にリーダーが以下のような手順でこれを構築する。

1) GitHub 上のメニュー New repository から、空のリモートリポジトリを作成する。
2) 空のリモートリポジトリをクローンして自分のマシン上にローカルリポジトリを作成する。
3) 必要な作業ファイルをローカルリポジトリの管理下に置いて、初回コミットを行う。
4) 中身が入ったローカルリポジトリを中央リポジトリにプッシュする

以上の操作で、中央 (リモート) リポジトリに初期のファイルがセットされることになる。

4.3. 共同編集者がそれぞれクローンを作成

GitHub は基本的にはファイルを管理する場所で、共同で直接編集を行う場所ではない。そこで共同編集メンバーは、まず中央リポジトリのクローンをローカルリポジトリとして手元に作成する (図 11)。

図 11. クローンの作成

4.4. 共同編集者による プル と プッシュ

メンバーはそれぞれ手元でファイルの編集・更新を行い、ローカルのコミットとリモートへのプッシュを進める。

ここで重要なことは「中央リポジトリを常に正の状態 (Positive State) に保つ」ということである。中央リポジトリは随時アップデートされており、手元のファイルが古くなっている可能性がある。したがって編集作業においては、事前にプル (リモートの変更をローカルに反映) することが必要である。自身の変更をプッシュ (ローカルの変更をリモートに反映) する場合も、まず最新の状態をプルして、自身の変更とマージ (併合) してから中央へプッシュ・・という手順になる。

プルできる状態、つまり中央の更新が先行しているにもかかわらず、プッシュしようとすると、警告が出る仕組みになっているため、中央のファイルは行き戻りなく更新される。中央リポジトリでは直接マージは行わない。あくまでローカルリポジトリで最新データをマージした上で作業を進めるのである。プッシュは基本的に中央の状態を前進させる場合にのみ適用される。

ローカルの変更をリモートにプッシュする様子を図 12 に示す。ahead という表記は、ローカルの作業がリモートに先行していることを意味し、この状態でプッシュが可能である。プッシュが反映されると、GitHub 上でもコミット単位での変更点が確認できる。

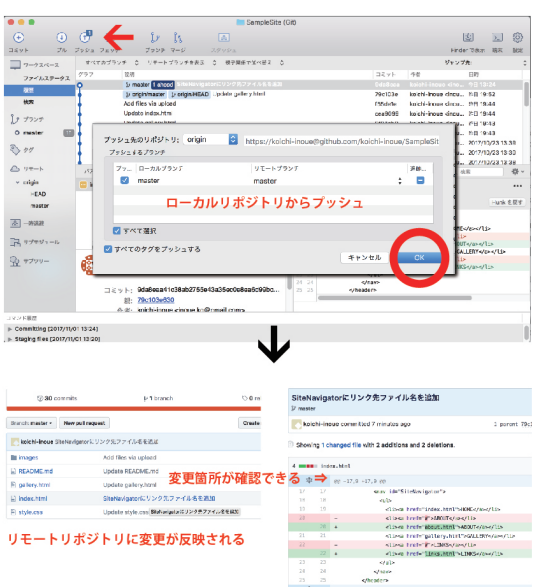


図 12. ローカルからリモートへのプッシュ

4.5. プルリクエスト (Pull Request)

GitHub のサービスにおいて、最も注目されているのが Pull Request である。これはローカルリポジトリでの変更を他の開発者に通知する機能で、機能追加や改修などの作業内容を関係者に通知するかたちで、共同編集者間のコミュニケーションが生まれる。ソーシャルコーディングの要となる機能である。

4.6. フォーク (Fork)

フォークとは他のユーザーのリポジトリを自分のアカウント内に複製することを言う。任意のユーザに対し、以下のような手順でその開発に貢献、あるいは派生版を作成することができる。

1) 既存のリポジトリをフォーク
2) フォークしたリポジトリをローカルにクローン
3) ローカルリポジトリ内で開発作業を行う
4) 更新を、フォークしたリポジトリにプッシュ
5) オリジナルの開発に貢献する場合は、オリジナルリポジトリにプルリクエストを送信

フォーク (複製) したリポジトリは自分の所有物なので、自由に更新ができる (従来の Copyright とは異なる OSS 開発に特有の発想)。ここから派生した別のプロジェクトが動き出すことも多く、実際 OSS には多くの派生版が存在する。

4.7. 課題管理 (Issue)

プログラムのバグや機能追加の要望などを管理するツールとして Issue という一種の SNS 機能がある。ここには誰でも自由にコメントや画像を投稿することができるため、開発に関わる議論をわかりやすく可視化することができる。

組織によっては、これをディレクター、デザイナー、営業スタッフも含めて活用している。

4.8. GitHub ページ

GitHub リポジトリから直接 Web サイトをホストする機能として GitHub Pages がある。開発中のリポジトリに HTML、CSS、JavaScript で構成された静的なウェブサイトがある場合、これを GitHub サブドメインでホストすることができる。更新と同時にチェックができる点で Web 制作のプロジェクトにとっては特筆すべき機能である。

5. GitHub の活用事例

この節では、ソーシャルコーディングサービスとしての GitHub の活用事例を紹介し、その可能性について考察する。

5.1. OSS 開発

オープンソースソフトウエア (OSS) の共同開発は GitHub の主目的である。Linux 関連のコードをはじめとして、Web アプリケーションフレームワークの Bootstrap、アイコンフォント集 Font Awesome、JavaScript の定番ライブラリ jQuery、テキストエディタの Atom など、無数にある状態で、Apache や Eclipse といった老舗のオープンソースプロジェクトもミラーリポジトリを GitHub で提供するようになっている。

5.2. Web デザイン

従来一般の Web デザイナーにはバージョン管理システムは敷居の高いツールであったが、GitHub の登場以来、その利用者が急増している。

メンバーごとに担当するファイルが異なる作業では衝突の発生もなく、またマークアップ言語の性質上、同一ファイルの共同編集の場合も致命的な衝突は発生しづらい。単なるフォルダの共有とは異なるバージョン管理の利用で生産効率は大幅に向上しているといえる。

5.3. 雑誌 (記事) の編集

GitHub では Web 上でファイルの作成・編集・ブランチの作成など一通りの操作が可能なため、Git そのものの知識のないライターでも簡単にファイルのバージョン管理ができる。

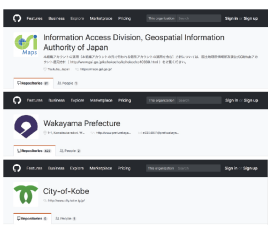
例えば、ライターが GitHub 上で記事を書き、執筆が完了した時点で PullRequst を編集者に送信する。編集者は PullRequst 上で記事を編集してマージする。編集前後の差分が残るため、ライターはそれをもとに記事を改善することもできるようになる。

例えば、ビジネスパーソンのための情報配信を行う SELECK では、記事の履歴管理に GitHub を導入し、企業のベストプラクティス、ツール情報、Tips などを配信している。

5.4. 公的機関のファイル配信

自治体のオープンデータへの取り組みが広がる中で、情報公開の手段として GitHub を活用する動きも始まっている。

国土地理院における地図情報の公開と更新 [*3]、和歌山県 [*4] や神戸市 [*5] によるオープンデータの取り組みは大きな話題となり、以後、同様の取り組みに着手する組織が増えている。



現在これらのデータは GitHub 上で誰でも自由にアクセスすることができる (図 13)。

図 13. 公的機関によるデータのオープン化

5.5. 研究・教育分野、その他での利用

研究・教育分野でも GitHub の活用がはじまっている。2016 年に科学雑誌ネイチャーに掲載された " Democratic databases: science on GitHub" という記事 [*6] でも、科学的なデータとコードの共有、維持、更新の可能性が評価されている。

また、GitHub は未来の開発者の育成についても意欲的に取り組んでおり、学生・教育機関向け制度として GitHub Education を開設している。

その他、求人活動 (就職活動) における GitHub の活用もある。エンジニア系の場合、GitHub がそのエンジニアを評価するのに有益な情報となるため、アカウントを登録するだけでエントリー完了する企業も登場してきた [*7]。

5.6. 筆者個人の活用事例

筆者自身も 2017 年から Web 系の演習科目において、GitHub によるサンプルソースコードの提供をはじめた [*8]。以前はホームページ上にソースを掲載する形で資料を提供していたが、GitHub 上では直接コードの編集とコミットを行うことができることと、更新されたファイルを即ダウンロードできる点で、教師卓で行う実演解説と確認作業の効率が大幅にアップした。今後は学生自身が GitHub による共同編集を体験できるような授業・プロジェクトの構築を検討している。

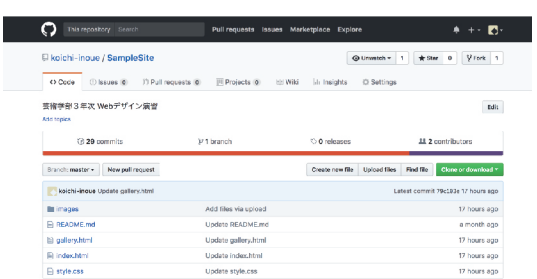


図 14. 筆者のアカウント｜演習用ソースコードの提供

6. まとめ

文書というものは、日々変わる状況に対応して更新を続けねばばらないと同時に、関係者全員で共有されねばならない。一方で我々の日常業務は日々増える作業項目と蛸壺化したフォルダの管理に追われ、混乱を極めるばかりである。

しかし 2008 年に登場した「GitHub - ソーシャルコーディング」は、OSS 開発の枠を超えた様々な文書の共有と公開を促すツールとして、一種のパラダイムシフトを実現しつつある。

V. パパネックが指摘するとおり [*9]、旧態依然とした著作権の考え方は、社会を萎縮させ、アイデアの拡散を停滞させる。今日あらゆる情報技術の基盤となっているオープンソースは、文字通り情報をオープンにすることによって、実り豊かな情報環境をつくった。公開して問題のない文書はオープンな場で共同編集する。分散リポジトリの考え方を多くの人が共有すれば、人と社会の未来はもう少し風通しが良くなるように思う。

もうひとつ大きなメリットがある。ソーシャルコーディングでは、どれだけ多くの仕事をしても物理的なゴミを一切出さない。

註

1) Google トレンド　https://trends.google.co.jp/trends/
2) 東洋経 ON LINE　http://toyokeizai.net/articles/-/137251
3) 国土地理院 GSI Maps　https://github.com/gsi-cyberjapan
4) 和歌山県 GitHub　https://github.com/wakayama-pref-org
5) 神戸市 GitHub　https://github.com/City-of-Kobe
6) Jeffrey Perke, Democratic databases: science on GitHub, Nature, 2016
7) ワンクリック採用｜面白法人カヤック https://www.kayac.com/recruit/
8) Koichi-Inoue GitHub　https://github.com/koichi-inoue
9) V. パパネック , 生きのびるためのデザイン , 晶文社 , 1974, p.11